

Triton

Generated by Doxygen 1.7.3

Tue Jan 27 2015 10:10:01

Contents

1	Triton(tm) Software Development Kit User's Manual	1
2	Obtaining a license for Triton, or how to evaluate it for free	3
3	Triton System Requirements	5
4	Getting started with Triton	7
4.1	Overview of the sample projects	7
4.2	Configuring your project	7
4.2.1	Linking with Triton under C++ and Windows	7
4.2.2	Linking with Triton using C#	8
4.2.3	Linking with Triton under Linux	9
4.2.4	Using Triton's headers	9
4.3	Intializing Triton	10
4.4	Rendering each frame	12
4.4.1	Updating Triton's lighting conditions	12
4.4.2	Updating Triton's camera matrices	12
4.4.3	Rendering the Ocean	13
4.4.4	Rendering User-Defined Patches of Geometry	13
4.5	Integrating environment cube maps with Triton	15
4.6	Shutting down	15
5	Simulating specific sea conditions with Triton	17
5.1	Choosing a wave spectrum model	17
5.2	Specifying fetch lengths	18
5.3	Simulating swells	18
5.4	Simulating specific Beaufort or Douglas sea states	19
6	Simulating ship wakes with Triton	21
7	Spray effects and breaking waves	23
8	Rotor wash effects	25
9	Simulating impacts on the water	27
10	Applying decals to the water surface	29
11	Integrating Triton with terrain and shallow water	31

12 Synchronizing Triton across multiple viewports or channels	35
13 Intersection tests with Triton	37
14 Underwater rendering with Triton	39
15 Using Triton with No Rendering (Physics Only)	41
16 Performance tuning tips	43
17 Troubleshooting tips	45
17.1 I see aliasing or shimmering on the ocean.	45
17.2 I can't link the Triton libraries	45
17.3 My Ocean is black!	45
17.4 My water is all sorts of strange colors!	46
17.5 Nothing's showing up at all!	46
17.6 My ocean looks garbled or corrupt.	46
17.7 Triton crashed on me	47
18 Redistributing Triton with your application	49
19 Obtaining support	51
20 Advanced topics	53
20.1 Integrating with your own resource manager	53
20.2 Integrating with your own memory manager	53
20.3 Adding your own effects to the water	54
20.4 Building Triton from source	55
20.5 Integrating planar reflection maps with Triton	55
20.6 Restricting Use of Computing Resources	57
20.7 Using linear color space with Triton	58
21 Third-party license notices	59
21.1 FFTSS: A Fast Fourier Transform Library	59
21.2 Intel's Integrated Performance Primitives	60
21.3 Cgtextures.com	60
21.4 AMD Accelerated Parallel Processing Math Libraries	60
22 Class Documentation	63
22.1 Triton::Allocator Class Reference	63
22.1.1 Detailed Description	64
22.1.2 Member Function Documentation	64
22.1.2.1 SetAllocator	64
22.2 Triton::BreakingWavesParameters Class Reference	64
22.2.1 Detailed Description	66
22.2.2 Constructor & Destructor Documentation	66
22.2.2.1 BreakingWavesParameters	66
22.2.3 Member Function Documentation	66
22.2.3.1 SetAmplitude	66
22.2.3.2 SetAutoWaveDirection	67
22.2.3.3 SetDepthFalloff	67

22.2.3.4	SetSteepness	67
22.2.3.5	SetSteepnessVariance	67
22.2.3.6	SetSurgeDepth	67
22.2.3.7	SetWaveDirection	67
22.2.3.8	SetWavelength	68
22.2.3.9	SetWavelengthVariance	68
22.3	Triton::Environment Class Reference	68
22.3.1	Detailed Description	75
22.3.2	Constructor & Destructor Documentation	75
22.3.2.1	Environment	75
22.3.2.2	~Environment	75
22.3.3	Member Function Documentation	76
22.3.3.1	AddSwell	76
22.3.3.2	AddWindFetch	76
22.3.3.3	ClearSwells	76
22.3.3.4	ClearWindFetches	77
22.3.3.5	CullSphere	77
22.3.3.6	EnableOpenMP	77
22.3.3.7	GetAboveWaterVisibility	77
22.3.3.8	GetAmbientLightColor	78
22.3.3.9	GetBelowWaterVisibility	78
22.3.3.10	GetBreakingWavesParameters	78
22.3.3.11	GetCameraMatrix	78
22.3.3.12	GetCameraPosition	78
22.3.3.13	GetConfigOption	78
22.3.3.14	GetCoordinateSystem	79
22.3.3.15	GetDevice	79
22.3.3.16	GetDirectionalLightColor	79
22.3.3.17	GetEnvironmentMap	79
22.3.3.18	GetEnvironmentMapMatrix	79
22.3.3.19	GetHeightMap	79
22.3.3.20	GetHeightMapMatrix	80
22.3.3.21	GetLightDirection	80
22.3.3.22	GetMaximumWaveHeight	80
22.3.3.23	GetOpenMPEnabled	80
22.3.3.24	GetPlanarReflectionDisplacementScale	80
22.3.3.25	GetPlanarReflectionMap	80
22.3.3.26	GetPlanarReflectionMapMatrix	80
22.3.3.27	GetProjectionMatrix	81
22.3.3.28	GetRandomNumberGenerator	81
22.3.3.29	GetRenderer	81
22.3.3.30	GetResourceLoader	81
22.3.3.31	GetRightVector	81
22.3.3.32	GetSeaLevel	81
22.3.3.33	GetSunIntensity	81
22.3.3.34	GetUpVector	82
22.3.3.35	GetViewport	82
22.3.3.36	GetWind	82
22.3.3.37	GetWorldUnits	83
22.3.3.38	GetZoomLevel	83

22.3.3.39	Initialize	83
22.3.3.40	IsDirectX	84
22.3.3.41	IsGeocentric	84
22.3.3.42	IsOpenGL	84
22.3.3.43	SetAboveWaterVisibility	84
22.3.3.44	SetAmbientLight	84
22.3.3.45	SetBelowWaterVisibility	84
22.3.3.46	SetBreakingWavesParameters	85
22.3.3.47	SetCameraMatrix	85
22.3.3.48	SetConfigOption	85
22.3.3.49	SetDirectionalLight	86
22.3.3.50	SetDouglasSeaScale	86
22.3.3.51	SetEnvironmentMap	86
22.3.3.52	SetHeightMap	87
22.3.3.53	SetLicenseCode	88
22.3.3.54	SetPlanarReflectionMap	88
22.3.3.55	SetProjectionMatrix	89
22.3.3.56	SetRandomNumberGenerator	90
22.3.3.57	SetSeaLevel	90
22.3.3.58	SetSunIntensity	90
22.3.3.59	SetViewport	90
22.3.3.60	SetWorldUnits	91
22.3.3.61	SetZoomLevel	91
22.3.3.62	SimulateSeaState	91
22.3.3.63	TRITON_VECTOR	92
22.4	Triton::Impact Class Reference	92
22.4.1	Detailed Description	93
22.4.2	Constructor & Destructor Documentation	94
22.4.2.1	Impact	94
22.4.3	Member Function Documentation	94
22.4.3.1	GetPosition	94
22.4.3.2	GetVelocity	94
22.4.3.3	Trigger	94
22.5	Triton::Matrix3 Class Reference	95
22.5.1	Detailed Description	97
22.5.2	Constructor & Destructor Documentation	97
22.5.2.1	Matrix3	97
22.5.2.2	Matrix3	97
22.5.2.3	Matrix3	97
22.5.2.4	~Matrix3	98
22.5.3	Member Function Documentation	98
22.5.3.1	FromRx	98
22.5.3.2	FromRy	98
22.5.3.3	FromRz	98
22.5.3.4	FromXYZ	98
22.5.3.5	operator*	98
22.5.3.6	operator*	98
22.5.3.7	ToFloatArray	98
22.5.3.8	Transpose	98
22.5.4	Friends And Related Function Documentation	99

22.5.4.1	operator*	99
22.6	Triton::Matrix4 Class Reference	99
22.6.1	Detailed Description	101
22.6.2	Constructor & Destructor Documentation	101
22.6.2.1	Matrix4	101
22.6.2.2	Matrix4	101
22.6.2.3	Matrix4	101
22.6.2.4	~Matrix4	102
22.6.3	Member Function Documentation	102
22.6.3.1	GetRow	102
22.6.3.2	InverseCramers	102
22.6.3.3	operator*	102
22.6.3.4	operator*	102
22.6.3.5	operator*	102
22.6.3.6	ToFloatArray	102
22.6.3.7	Transpose	102
22.6.4	Friends And Related Function Documentation	102
22.6.4.1	operator*	102
22.7	Triton::MemObject Class Reference	103
22.7.1	Detailed Description	105
22.8	Triton::Ocean Class Reference	105
22.8.1	Detailed Description	110
22.8.2	Constructor & Destructor Documentation	110
22.8.2.1	~Ocean	110
22.8.3	Member Function Documentation	111
22.8.3.1	AddDecal	111
22.8.3.2	ComputeReflectionMatrices	111
22.8.3.3	Create	112
22.8.3.4	Create	113
22.8.3.5	D3D9DeviceLost	114
22.8.3.6	D3D9DeviceReset	114
22.8.3.7	Draw	115
22.8.3.8	EnableGodRays	115
22.8.3.9	EnableSpray	115
22.8.3.10	EnableWireframe	115
22.8.3.11	GetChoppiness	116
22.8.3.12	GetDepth	116
22.8.3.13	GetDepthOffset	116
22.8.3.14	GetDisplacementDampingDistance	116
22.8.3.15	GetEnvironment	116
22.8.3.16	GetFFTName	117
22.8.3.17	GetGodRaysFade	117
22.8.3.18	GetHeight	117
22.8.3.19	GetIntersection	118
22.8.3.20	GetLinearColorSpace	118
22.8.3.21	GetLoopingPeriod	118
22.8.3.22	GetNumTriangles	118
22.8.3.23	GetPlanarReflectionBlend	118
22.8.3.24	GetQuality	119
22.8.3.25	GetRefractionColor	119

22.8.3.26	GetShaderObject	119
22.8.3.27	GetWaveHeading	119
22.8.3.28	GodRaysEnabled	120
22.8.3.29	IsCameraAboveWater	120
22.8.3.30	Lock	120
22.8.3.31	ReloadShaders	120
22.8.3.32	RemoveDecal	120
22.8.3.33	ScaleDecal	120
22.8.3.34	SetChoppiness	120
22.8.3.35	SetDecalAlpha	121
22.8.3.36	SetDepth	121
22.8.3.37	SetDepthOffset	121
22.8.3.38	SetDisplacementDampingDistance	122
22.8.3.39	SetGodRaysFade	122
22.8.3.40	SetLinearColorSpace	122
22.8.3.41	SetLoopingPeriod	122
22.8.3.42	SetPatchMatrix	122
22.8.3.43	SetPatchShader	123
22.8.3.44	SetPlanarReflectionBlend	124
22.8.3.45	SetQuality	124
22.8.3.46	SetRefractionColor	124
22.8.3.47	SprayEnabled	125
22.8.3.48	Unlock	125
22.8.3.49	UnsetPatchShader	125
22.8.3.50	UpdateSimulation	125
22.9	Triton::OrientedBoundingBox Class Reference	126
22.9.1	Detailed Description	127
22.9.2	Constructor & Destructor Documentation	128
22.9.2.1	OrientedBoundingBox	128
22.9.3	Member Function Documentation	128
22.9.3.1	PointInBox	128
22.9.3.2	Set	128
22.10	Triton::RandomNumberGenerator Class Reference	128
22.10.1	Detailed Description	129
22.10.2	Member Function Documentation	129
22.10.2.1	GetRandomDouble	129
22.10.2.2	GetRandomInt	130
22.10.2.3	SetRandomSeed	130
22.11	Triton::ResourceLoader Class Reference	130
22.11.1	Detailed Description	132
22.11.2	Constructor & Destructor Documentation	132
22.11.2.1	ResourceLoader	132
22.11.3	Member Function Documentation	132
22.11.3.1	FreeResource	132
22.11.3.2	LoadResource	133
22.11.3.3	SetResourceDirPath	133
22.12	Triton::RotorWash Class Reference	134
22.12.1	Detailed Description	135
22.12.2	Constructor & Destructor Documentation	135
22.12.2.1	RotorWash	135

22.12.3 Member Function Documentation	135
22.12.3.1 GetPosition	135
22.12.3.2 GetVelocity	136
22.12.3.3 Update	136
22.13 Triton::SwellDescription Class Reference	136
22.13.1 Detailed Description	138
22.14 Triton::TidalStreamWake Class Reference	138
22.14.1 Detailed Description	139
22.14.2 Constructor & Destructor Documentation	139
22.14.2.1 TidalStreamWake	139
22.14.3 Member Function Documentation	140
22.14.3.1 Update	140
22.15 Triton::Utils Class Reference	140
22.15.1 Detailed Description	142
22.15.2 Member Function Documentation	142
22.15.2.1 ClearGLErrors	142
22.15.2.2 GetDecalShaderFileName	142
22.15.2.3 GetDLLExtension	142
22.15.2.4 GetDLLPath	142
22.15.2.5 GetDLLSuffix	142
22.15.2.6 GetDX9Macros	143
22.15.2.7 GetGodRayShaderFileName	143
22.15.2.8 GetParticleShaderFileName	143
22.15.2.9 GetWaterShaderFileName	143
22.15.2.10 PrintGLErrors	143
22.15.2.11 SetDebugMsgCB	143
22.16 Triton::Vector3 Class Reference	144
22.16.1 Detailed Description	146
22.16.2 Constructor & Destructor Documentation	146
22.16.2.1 Vector3	146
22.16.2.2 Vector3	146
22.16.3 Member Function Documentation	146
22.16.3.1 Cross	146
22.16.3.2 Dot	146
22.16.3.3 Length	146
22.16.3.4 Normalize	146
22.16.3.5 operator!=	146
22.16.3.6 operator*	147
22.16.3.7 operator*	147
22.16.3.8 operator+	147
22.16.3.9 operator+	147
22.16.3.10 operator-	147
22.16.3.11 operator==	147
22.16.3.12 Serialize	147
22.16.3.13 SquaredLength	147
22.16.3.14 Unserialize	147
22.16.4 Member Data Documentation	148
22.16.4.1 x	148
22.17 Triton::Vector3f Class Reference	148
22.17.1 Detailed Description	149

22.17.2 Constructor & Destructor Documentation	150
22.17.2.1 Vector3f	150
22.17.2.2 Vector3f	150
22.17.2.3 Vector3f	150
22.17.3 Member Function Documentation	150
22.17.3.1 Dot	150
22.17.3.2 Length	150
22.17.3.3 Normalize	150
22.17.3.4 operator*	150
22.17.3.5 operator*	150
22.17.3.6 operator+	150
22.17.3.7 operator-	151
22.17.4 Member Data Documentation	151
22.17.4.1 x	151
22.18 Triton::Vector4 Class Reference	151
22.18.1 Detailed Description	153
22.18.2 Constructor & Destructor Documentation	153
22.18.2.1 Vector4	153
22.18.2.2 Vector4	153
22.18.3 Member Function Documentation	153
22.18.3.1 Dot	153
22.18.3.2 operator*	153
22.18.3.3 operator*	153
22.18.3.4 operator+	153
22.18.3.5 operator+	154
22.18.3.6 operator-	154
22.18.4 Member Data Documentation	154
22.18.4.1 x	154
22.19 Triton::WakeGenerator Class Reference	154
22.19.1 Detailed Description	156
22.19.2 Constructor & Destructor Documentation	156
22.19.2.1 WakeGenerator	156
22.19.3 Member Function Documentation	157
22.19.3.1 ClearWakes	157
22.19.3.2 GetLODDistance	157
22.19.3.3 GetParameters	157
22.19.3.4 GetPosition	157
22.19.3.5 GetSternPosition	157
22.19.3.6 GetVelocity	157
22.19.3.7 HasPropWash	157
22.19.3.8 SetLODDistance	158
22.19.3.9 SetParameters	158
22.19.3.10 Update	158
22.20 Triton::WakeGeneratorParameters Class Reference	158
22.20.1 Detailed Description	161
22.20.2 Constructor & Destructor Documentation	161
22.20.2.1 WakeGeneratorParameters	161
22.20.3 Member Data Documentation	161
22.20.3.1 bowSize	161
22.20.3.2 bowSprayOffset	162

22.20.3.3	bowWaveOffset	162
22.20.3.4	draft	162
22.20.3.5	numHullSprays	162
22.20.3.6	propWashOffset	162
22.20.3.7	sprayVelocityScale	162
22.21	Triton::WindFetch Class Reference	162
22.21.1	Detailed Description	164
22.21.2	Constructor & Destructor Documentation	164
22.21.2.1	WindFetch	164
22.21.3	Member Function Documentation	164
22.21.3.1	ClearFetchLength	164
22.21.3.2	ClearLocalization	164
22.21.3.3	GetWindAtLocation	165
22.21.3.4	SetFetchLength	165
22.21.3.5	SetLocalization	165
22.21.3.6	SetWind	166
23	File Documentation	167
23.1	C:/triton/trunk/Public Headers/Environment.h File Reference	167
23.1.1	Detailed Description	169
23.2	C:/triton/trunk/Public Headers/Impact.h File Reference	169
23.2.1	Detailed Description	171
23.3	C:/triton/trunk/Public Headers/Matrix3.h File Reference	171
23.3.1	Detailed Description	172
23.4	C:/triton/trunk/Public Headers/Matrix4.h File Reference	172
23.4.1	Detailed Description	174
23.5	C:/triton/trunk/Public Headers/MemAlloc.h File Reference	174
23.5.1	Detailed Description	175
23.6	C:/triton/trunk/Public Headers/Ocean.h File Reference	176
23.6.1	Detailed Description	177
23.7	C:/triton/trunk/Public Headers/OrientedBoundingBox.h File Reference	177
23.7.1	Detailed Description	179
23.8	C:/triton/trunk/Public Headers/RandomNumberGenerator.h File Reference	179
23.8.1	Detailed Description	181
23.9	C:/triton/trunk/Public Headers/ResourceLoader.h File Reference	181
23.9.1	Detailed Description	183
23.10	C:/triton/trunk/Public Headers/RotorWash.h File Reference	183
23.10.1	Detailed Description	185
23.11	C:/triton/trunk/Public Headers/TidalStreamWake.h File Reference	185
23.11.1	Detailed Description	186
23.12	C:/triton/trunk/Public Headers/Triton.h File Reference	186
23.12.1	Detailed Description	187
23.13	C:/triton/trunk/Public Headers/TritonCommon.h File Reference	187
23.13.1	Detailed Description	189
23.14	C:/triton/trunk/Public Headers/Vector3.h File Reference	189
23.14.1	Detailed Description	190
23.15	C:/triton/trunk/Public Headers/Vector4.h File Reference	190
23.15.1	Detailed Description	192
23.16	C:/triton/trunk/Public Headers/WakeGenerator.h File Reference	192

23.16.1 Detailed Description	194
23.17C:/triton/trunk/Public Headers/WindFetch.h File Reference	194
23.17.1 Detailed Description	195

Chapter 1

Triton(tm) Software Development Kit User's Manual

Thanks for using Triton from Sundog Software! Triton is a C++ and C# library for real-time water rendering, featuring easy integration with OpenGL 2.0+, DirectX9, and DirectX11 based Windows applications. It provides realistic visuals of waves for any given wind conditions or a specified Beaufort scale, and takes advantage of general-purpose GPU (GPGPU) computing on systems that support OpenCL, CUDA, or DirectX11 Compute Shaders (DirectCompute.) This allows us to compute Fast Fourier Transforms on thousands of waves at once at rates of over 100 frames per second on most modern systems.

Triton also features the ability to seamlessly integrate with geocentric coordinate systems in addition to flat coordinate systems. The surface of Triton's sea may be configured to conform to a WGS84 ellipsoid, or a spherical model of the Earth.

This manual will get you started - you should be surprised at how quickly you can be up and running. We also provide a detailed class reference for our public interfaces; licensed users of Triton will receive Triton's full source and full documentation of our internal classes as well.

- [Obtaining a license for Triton, or how to evaluate it for free](#)
- [Triton System Requirements](#)
- [Getting started with Triton](#)
 - [Overview of the sample projects](#)
 - [Configuring your project](#)
 - [Intializing Triton](#)
 - [Integrating environment cube maps with Triton](#)
 - [Rendering each frame](#)
 - [Shutting down](#)
- [Simulating specific sea conditions with Triton](#)

- [Simulating ship wakes with Triton](#)
- [Spray effects and breaking waves](#)
- [Rotor wash effects](#)
- [Simulating impacts on the water](#)
- [Applying decals to the water surface](#)
- [Integrating Triton with terrain and shallow water](#)
- [Intersection tests with Triton](#)
- [Underwater rendering with Triton](#)
- [Synchronizing Triton across multiple viewports or channels](#)
- [Using Triton with No Rendering \(Physics Only\)](#)
- [Performance tuning tips](#)
- [Troubleshooting tips](#)
- [Redistributing Triton with your application](#)
- [Obtaining support](#)
- [Advanced topics](#)
 - [Integrating with your own resource manager](#)
 - [Integrating with your own memory manager](#)
 - [Adding your own effects to the water](#)
 - [Building Triton from source](#)
 - [Integrating planar reflection maps with Triton](#)
 - [Restricting Use of Computing Resources](#)
 - [Using linear color space with Triton](#)
- [Third-party license notices](#)

Chapter 2

Obtaining a license for Triton, or how to evaluate it for free

License codes for Triton may be purchased online at <http://www.sundog-soft.com/> using a credit card; for other payment options, contact sales@sundog-soft.com.

Once you've received your license name and code, all you have to do is pass them into `Triton::Environment::SetLicenseCode()` after you've created your Environment object, and all restrictions on Triton will be removed.

We do make Triton freely available in an "evaluation mode" if you don't pass it a valid license code. This lets you confirm you can integrate Triton with your application before committing to a purchase. Without a license code, Triton will display a warning dialog at startup, and terminate your application after five minutes of runtime - but this should be sufficient to see if you can get Triton up and running with your project.

Our license terms are simple; a single price lets you distribute as many copies or channels of your executable, and incremental updates to it, that links in Triton. No royalties, no per-channel costs, no per-developer costs. Licensees also receive Triton's full source, 3 months of technical support, and free updates.

If you have any questions about Triton's licensing (or about anything, really,) contact us at sales@sundog-soft.com.

Chapter 3

Triton System Requirements

Although Triton is able to take advantage of the latest general purpose GPU technologies and graphics SDK's, it remains compatible with lower end systems as well.

Any system capable of running OpenGL 2.0, or DirectX9 with Shader Model 3.0, is supported. Triton may also be used with OpenGL 3.x, OpenGL 4.x, and DirectX 11. Support for DirectX9Ex is also provided.

It is possible to run Triton on older systems with Shader Model 2.0 pixel shaders and 3.0 vertex shaders, but typically these systems are not powerful enough to run Triton reliably.

If you encounter any trouble with compatibility or performance on a supported system profile, don't hesitate to contact us at support@sundog-soft.com.

Chapter 4

Getting started with Triton

Triton includes over 60,000 lines of code, but it will only take a few to integrate it into your application.

Here's how.

4.1 Overview of the sample projects

Examining the "Samples" directory of the SDK is a quick way to get started. You'll find simple applications illustrating the use of Triton in OpenGL, DirectX9, DirectX11, C#, OpenSceneGraph, and Ogre applications, with some handy functions for initializing, updating, and shutting down Triton that you can use in your own app. Each sample also includes a sky box class, which is used not just to make the sample look prettier, but also illustrates the integration of environment cube maps with Triton for more realistic reflections from the sky.

You'll find examples of using Triton to render infinite oceans, as well as smaller bodies of water using user-defined patches of geometry.

The OpenGL sample is built on OpenGL 2.0 functionality. However, the core code of the OpenGL sample avoids use of the fixed function pipeline, and should be illustrative for developers working under OpenGL 3, 4, and beyond as well.

If you are using osgEarth, getting started is even easier. osgEarth includes a simple driver for Triton that can get you up and running quickly.

4.2 Configuring your project

4.2.1 Linking with Triton under C++ and Windows

Triton provides libraries for win32 and x64 applications created with Microsoft Visual Studio 2005, 2008, 2010, 2012, or 2013 (with the latest service packs applied and security patches) for every runtime library flavor. You'll find the libraries in the lib

directory of the SDK. In your project properties, add the appropriate library to your linker inputs.

The Triton SDK installer defines the environment variable TRITON_PATH that you may use when referencing Triton's libraries and headers in your project properties.

For example, for a Win32 application developed with Visual Studio 2010 and using the "multi-threaded DLL" runtime, you'd link against

```
$(TRITON_PATH)/lib/vc10/win32/Triton-MT-DLL.lib.
```

Visual Studio 2003.NET libraries are found in "lib/vc7", Visual Studio 2005 libraries are found in "lib/vc8", Visual Studio 2008 libraries are found in "lib/vc9", Visual Studio 2010 libraries are in "lib/vc10", Visual Studio 2012 libraries are in "lib/vc11", and Visual Studio 2013 libraries are in "lib/vc12". Refer to the following table for matching the appropriate library file with the runtime your project is using (which you can find under the "C/C++ / Code Generation" property page in your project.)

Our Visual Studio 2012 and 2013 libraries are linked against the Windows SDK 8, and only support desktop applications at this time.

One special note for Visual Studio 2013 users - since the version of CUDA we use does not include Visual Studio 2013 support, our CUDA DLL for Visual Studio 2013 is built with Visual Studio 2012 tools. This means you may need to install Visual Studio 2012 express edition in order to get the necessary runtime DLL's on your system to develop with.

Runtime flavor	Triton library
Multi-threaded	Triton-MT.dll
Multi-threaded Debug	Triton-MTD.dll
Multi-threaded DLL	Triton-MT-DLL.dll
Multi-threaded Debug DLL	Triton-MTD-DLL.dll

Even though we provide specific builds for individual compilers and runtimes, some of the third party DLL's we incorporate do not. If you run into trouble linking while using runtimes other than multi-threaded DLL, try adding MSVCRT to the "Ignore specific default libraries" field of the "Linker / Input" property page of your project.

You may also experience runtime problems if your application is built with special flags for the standard runtime library, such as `_SECURE_SCL` or `_HAS_ITERATOR_DEBUGGING = 0`. If you have trouble linking either at compile or runtime, please contact support@sundog-soft.com. We can provide you with an SDK that includes obfuscated source code, allowing you to build Triton with whatever development environment and compile-time flags you need.

4.2.2 Linking with Triton using C#

Inside the Samples/CSharpSample folder, you'll find TritonDLL.dll. This is Triton's native code, packaged as a DLL that may be invoked from C# code. Deploy this DLL into the working directory of your C# application.

Also inside the CSharpSample, you'll find the TritonClassLibrary project. This is a C# wrapper over Triton's C++ API that will be your interface to Triton. Include this project in your solution, and reference it from your own app's project.

The TritonXNA project is a sample illustrating using Triton from a C# XNA Game Framework 4.0 application. Refer to the `readme.txt` file inside the `CSharpSample` folder for more details on how it works, and how to use Triton from your own C# project.

The `TritonClassLibrary` is automatically generated from our C++ API; you may refer to the C++ API reference in our docs folder for the C# classes - they work exactly the same way.

4.2.3 Linking with Triton under Linux

The evaluation version of Triton for Linux is distributed as obfuscated source code, which you will build against your own system. Licensed users receive unobfuscated source.

You will need to have the latest graphics drivers installed. Notably, the open-source ATI drivers provided with Ubuntu are, as of this writing, known to not properly support OpenCL/OpenGL interoperability which will cause Triton to fail on ATI cards. You'll need to uninstall the `fglrx` drivers and install the latest from AMD's website if this happens.

Building Triton for Linux requires several third-party development kits to be installed first:

CMake: <http://www.cmake.org/> FFTSS: <http://www.ssisc.org/fftss/download.html>
AMD's APP SDK (optional): <http://developer.amd.com/sdks/AMDAPPSDK/downloads/Pages/default.aspx>
NVIDIA's CUDA Toolkit (optional): <http://developer.nvidia.com/cuda-toolkit>
AMD's APPML (optional): <http://developer.amd.com/libraries/appmathlibs/Pages/default.aspx>
Intel's Integrated Performance Primitives (highly recommended): <http://software.intel.com/en-us/articles/intel-integrated-performance-primitives>

If you're running with an NVIDIA card, installing the CUDA toolkit may result in big performance gains. Similarly, AMD/ATI customers should install the AMD APP SDK.

Once the prerequisites are installed, run our installer script as super-user to build Triton on your system, along with the OpenGL sample application. If CMake gives you errors about missing variables such as `GLUT_XI_LIBRARY` or `GLUT_Xmu_LIBRARY`, you may need to install the following packages first:

```
sudo apt-get install libxmu-dev libxi-dev
```

Please don't hesitate to contact support@sundog-soft.com if you have trouble building Triton on your system.

4.2.4 Using Triton's headers

Under the "Additional include directories" field in your project's "C/C++" property page, add the following:

```
$(TRITON_PATH)/Public Headers
```

With this in place, you can simply

```
#include "Triton.h"
```

to gain access to Triton's capabilities.

C# developers will simply reference the TritonClassLibrary project in their own project, and place a

```
using Triton;
```

prior to referencing classes within the TritonClassLibrary.

4.3 Initializing Triton

There are three main objects you'll need to create at startup in order to use Triton.

The first thing you'll need is a [Triton::ResourceLoader](#). This class lets Triton access its shaders, DLL's, and texture resources, and it's what you'll use to tell Triton where these resources are located. The SDK includes a "resources" directory that you may redistribute, and you just need to provide an absolute or relative path to where you installed this directory in the [Triton::ResourceLoader](#)'s constructor. If you're interested in integrating Triton with your own resource manager, see [Integrating with your own resource manager](#).

Next, you'll need to create and initialize a [Triton::Environment](#) object, using the [Triton::ResourceLoader](#) you just made. The [Triton::Environment](#) class lets you specify the coordinate system, rendering system, and environmental conditions affecting Triton's water. For example, to integrate Triton with a DirectX11-based simulation application in a geocentric coordinate system using the WGS84 ellipsoid with the Z axis pointing through the poles, you'd call [Triton::Environment::Initialize\(\)](#) with the parameters [Triton::WGS84_ZUP](#) and [Triton::DIRECTX_11](#) as well as the ResourceLoader you created. You'll find support for flat-earth coordinate systems and spherical systems as well, with "up" on the Z or Y axes, and for OpenGL 2.x, 3.x, 4.x, DirectX9, and DirectX11. Be sure to check for error codes from [Triton::Environment::Initialize\(\)](#) - the most likely problem is that the resource path used to create your ResourceLoader isn't quite right.

If you have purchased a license for Triton, call [Triton::Environment::SetLicenseCode\(\)](#) to remove the evaluation restrictions on the SDK.

You'll then need to add some wind to Triton's simulation, or there won't be any waves. Create one or more [Triton::WindFetch](#) objects and add them via [Triton::Environment::AddWindFetch\(\)](#). A wind fetch represents a region of a specific wind speed and direction, which may be localized to an ellipsoidal area. If more than one wind fetch is present at a location, they'll be added together. Alternately, you can use the [Triton::Environment::SimulateSeaState\(\)](#) method to quickly simulate a given state on the Beaufort scale.

Finally, you'll create the [Triton::Ocean](#) object itself, using the [Triton::Environment](#) you've created. There are three [Triton::WaterModelTypes](#) you can select from when constructing an Ocean: TESSENDORF, JONSWAP, and PIERSON-MOSKOWITZ. All three use fast-Fourier transforms to simulate thousands of waves at once. TESSENDORF

uses the same algorithms used in feature films, while JONSWAP provides a more realistic wave spectrum that is a better fit for maritime training applications. PIERSON-MOSKOWITZ is similar to JONSWAP, but cannot handle the effects of wind "fetch length," or how far the wind has been travelling. Usually we recommend starting with JONSWAP.

If `Triton::Ocean::Create` returns a null pointer, see [Troubleshooting tips](#) to figure out what's going on.

A complete example of initializing Triton's objects follows.

```
// Create the Triton objects at startup, once we have a valid GL context in place

bool InitTriton()
{
    // We use the default resource loader that just loads files from disk. You'll
    // need
    // to redistribute the resources folder if using this. You can also extend the
    // ResourceLoader class to hook into your own resource manager if you wish.
    resourceLoader = new Triton::ResourceLoader("../..\\resources\\");

    // Create an environment for the water, with a flat-Earth coordinate system with Y
    // pointing up and using an OpenGL 2.0 capable context.
    environment = new Triton::Environment();
    Triton::EnvironmentError err = environment->Initialize(Triton::FLAT_YUP,
        Triton::OPENG_2_0, resourceLoader);
    if (err != Triton::SUCCEEDED) {
        ::MessageBoxA(NULL, "Failed to initialize Triton - is the resource path passed in to "
            "the ResourceLoader constructor valid?", "Triton error", MB_OK | MB_ICONEXCLAMATION);
        return false;
    }

    // Substitute your own license name and code, otherwise the app will terminate after
    // 5 minutes. Visit www.sundog-soft.com to purchase a license if you're so inclined.
    environment->SetLicenseCode("Your license name", "Your license code");

    // Set up wind of 10 m/s blowing North
    Triton::WindFetch wf;
    wf.SetWind(10.0, 0.0);
    environment->AddWindFetch(wf);

    // Finally, create the Ocean object using the environment we've created.
    // If NULL is returned, something's wrong - enable the enable-debug-messages option
    // in resources/triton.config to get more details on the problem.
    ocean = Triton::Ocean::Create(environment, Triton::TESSENDORF);

    return (ocean != NULL);
}
```

4.4 Rendering each frame

If your camera includes a region of your scene containing water, you'll need to render your Ocean object as part of your scene. Doing involves three steps:

4.4.1 Updating Triton's lighting conditions

Call `Triton::Environment::SetDirectionalLight()` to specify the color and direction of sunlight (or moonlight.) This information will be used to create specular reflections of the sun or moon in the water. Note this is the direction **to** the sun or moon, not from it - getting the direction wrong will lead to invalid coloration of the water.

`Triton::Environment::SetAmbientLight()` provides the ambient skylight used to light the seafoam and the water itself, if not environment map is provided (see [Integrating environment cube maps with Triton](#))

You might use Triton in conjunction with a system that provides scene lighting from dynamic time of day effects, such as Sundog Software's SilverLining library (see <http://www.sundog-soft.com/>) Or, these lighting values might be derived from the skybox texture you're using.

For example:

```
// Position the sun 45 degrees up in the sky at full brightness:
Triton::Vector3 lightPosition(0, 1.0 / sqrt(2.0), 1.0 / sqrt(2.0));
environment->SetDirectionalLight(lightPosition, Triton::Vector3(1.0, 1.0, 1.0));

// Ambient color of the sky:
environment->SetAmbientLight(Triton::Vector3(0.6, 0.9, 0.9));
```

4.4.2 Updating Triton's camera matrices

Since Triton does not rely on fixed function pipelines, you'll need to tell it explicitly what your camera matrices are so we may render the ocean consistently with the rest of your scene. Just pass in the modelview and projection matrices for your scene using `Triton::Environment::SetCameraMatrix()` and `Triton::Environment::SetProjectionMatrix()`. Both methods take in an array of 16 doubles. For example:

```
double projection[16];
double modelview[16];

// How you retrieve your camera matrices will vary depending on the engine
// you're using, so we'll just postulate the existence of these methods:
GetProjectionMatrix(projection);
GetModelviewMatrix(modelview);

environment->SetCameraMatrix(modelview);
environment->SetProjectionMatrix(projection);
```

Engines sometimes vary on whether they expose row-major or column-major matrices. If your ocean isn't rendering properly, try transposing the matrix before passing it in. If all else fails, try constructing these matrices from scratch using the camera's position,

field of view, clip planes, and aspect ratio. Refer to the sample code included with Triton for examples.

Passing in your matrices in C# is a little tricky. Here's an example of creating a projection matrix in XNA and passing it into Triton; use the same technique for the camera matrix.

```
projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4, GraphicsDevice.Viewport.AspectRatio, 10.0f, 100000.0f);

double[] m = new double[16];

m[0] = projection.M11; m[1] = projection.M12; m[2] = projection.M13; m[3] = projection.M14;
m[4] = projection.M21; m[5] = projection.M22; m[6] = projection.M23; m[7] = projection.M24;
m[8] = projection.M31; m[9] = projection.M32; m[10] = projection.M33; m[11] = projection.M34;
m[12] = projection.M41; m[13] = projection.M42; m[14] = projection.M43; m[15] = projection.M44;

SWIGTYPE_p_double matrix4 = TritonEnvironment.new_double_array(16);
for (int i = 0; i < 16; i++)
{
    TritonEnvironment.double_array_setitem(matrix4, i, m[i]);
}
environment.SetProjectionMatrix(matrix4);
TritonEnvironment.delete_double_array(matrix4);
```

4.4.3 Rendering the Ocean

With the lighting and camera information in place, we can now draw our ocean. Just call `Triton::Ocean::Draw()` and you're done. For example:

```
// Draw the ocean for the current time sample
if (ocean) {
    DWORD millis = timeGetTime();
    ocean->Draw((double)millis * 0.001);
}
```

Note that an explicit time sample is passed in, so you may manipulate the passage of time however you wish. Since the ocean may draw transparent spray particles, you'll usually want to call `Triton::Ocean::Draw()` at the end of your frame for proper sorting. If you need to draw the ocean surface and the particles separately, you'll find parameters on `Ocean::Draw()` to let you do that.

`Ocean::Draw()` renders an infinite ocean; to draw user-defined patches of water geometry, see [Rendering User-Defined Patches of Geometry](#).

4.4.4 Rendering User-Defined Patches of Geometry

In addition to drawing infinite oceans with the `Triton::Ocean::Draw()` method, Triton may also be used to shade your own water geometry. This allows you to use Triton for

smaller bodies of water. When used together with [Triton::Ocean::SetDepth\(\)](#), you may use Triton to render ponds, lakes, and water of any size and depth.

The [OpenGLPatchSample](#), [DirectX9PatchSample](#), and [DirectX11PatchSample](#) illustrate this technique. You'll use the [Triton::Ocean::SetPatchShader\(\)](#) method to set up the state and shaders required to draw Triton's water, then draw a flat mesh where you want your water to appear, and finally call [Triton::Ocean::UnsetPatchShader\(\)](#) to restore the previous state.

As you'll be doing your own drawing, you'll need to take care that the proper culling state is in place for your water patch. Also ensure that your depth test is set to "less than or equal" for best results.

Here's an example of rendering user-defined water geometry in OpenGL:

```
// Explicitly update the ocean simulation once per frame
environment->SetSeaLevel(5);
ocean->UpdateSimulation(time);

// Bind our vertex and index arrays to draw our mesh
glBindBuffer(GL_ARRAY_BUFFER, vboID);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, idxID);
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(4, GL_FLOAT, 0, 0);
glDepthFunc(GL_LEQUAL);

// Have Triton set all the state required to render our mesh as water
// with floating-point vertices with a stride of 16 bytes.
// We optionally pass in a 4x4 model matrix to translate the patch
ocean->SetPatchShader(time, 16, 0, false, modelMatrix);

// Draw our mesh - Triton will make it look like water...
glEnable(GL_CULL_FACE);
glFrontFace(GL_CW);
glDrawElements(GL_TRIANGLE_STRIP, nIndices, GL_UNSIGNED_INT, 0);

// Restore the previous state.
ocean->UnsetPatchShader();
```

Additional water patches per frame may be drawn in the same manner, but be sure that [Ocean::UpdateSimulation\(\)](#) is only called once per frame to preserve performance. You may call [Ocean::UpdateSimulation\(\)](#) from another thread if you wish. Also, take care that every call to [Ocean::SetPatchShader\(\)](#) is balanced with a call to [Ocean::UnsetPatchShader\(\)](#).

If you are drawing many patches in one scene, it will be most efficient to call [Ocean::SetPatchShader\(\)](#) once prior to drawing all of them, and then call [Ocean::SetPatchMatrix\(\)](#) to position each individual mesh. This allows you to avoid the overhead of [SetPatchShader\(\)](#) on every individual mesh, when all you need to do is change the position for each one.

Refer to the sample code provided for DirectX for more examples. Drawing user-defined geometry is supported in geocentric and flat-Earth coordinate systems, but only works with Tessendorf waves.

C# users will want to refer to [Updating Triton's camera matrices](#) for an example of passing a 4x4 matrix from C# into Triton, if you need to pass a translation matrix in to [Ocean::SetPatchShader\(\)](#). The [TritonOcean](#) class in the [TritonClassLibrary](#) also includes the helper functions for manipulating double arrays used in that example with [TritonEnvironment](#).

4.5 Integrating environment cube maps with Triton

By default, Triton will just reflect whatever color you passed in with [Triton::Environment::SetAmbientLight\(\)](#) in the water. For more realistic reflections, you'll want to set an environmental cube map using [Triton::Environment::SetEnvironmentMap\(\)](#). Doing so is optional, but it makes a big difference.

The type of texture parameter passed into this method will vary depending on what renderer you're using. OpenGL users should pass in a GLuint representing the texture ID of the cube map. DirectX9 users should pass in a LPDIRECT3DCUBETEXTURE9. (C# users may obtain this from the pComPtr member of their texture object.) DirectX11 users should pass a ID3D11ShaderResourceView pointer.

If you find that reflections seem to be coming from the wrong face of your cube map, you can correct for this using the Matrix3 parameter passed into [Environment::SetEnvironmentMap\(\)](#). This is especially common with DirectX due to the left-handed convention of DirectX cube maps. For example, if you seem to be getting reflections from the bottom of your environment map instead of from the top, and your "up" direction is the positive Y axis, pass in a scaling matrix to scale Y by -1.

Examples of loading and constructing properly constructed cube maps from disk may be found in the SkyBox classes in the SDK's sample code. You might also dynamically generate these cube maps based on a physical simulation of the sky such as Sundog Software's SilverLining library. The demo application for Triton found on our website does exactly that.

4.6 Shutting down

At shutdown time, C++ users just delete the Triton objects in the reverse order in which they were created - first your [Triton::Ocean](#), then your [Triton::Environment](#), and finally your [Triton::ResourceLoader](#). We'll clean up our memory and resources. For example:

```
// Clean up our resources
void Destroy()
{
    if (ocean) delete ocean;
    if (environment) delete environment;
    if (resourceLoader) delete resourceLoader;
}
```


Chapter 5

Simulating specific sea conditions with Triton

If you're using Triton for training and simulation purposes, you'll want to use its more advanced physical simulation capabilities.

There are several ways to specify ocean conditions with Triton.

One thing to understand first is the distinction between local wind waves and swells. Wind waves result from local wind conditions, as defined by the `WindFetch` objects that cover the observer. Swells are typically longer-wavelength waves that may have originated from distant storms not impacting the immediate area. You may want to specify these types of waves independently, and Triton lets you do that.

Another important concept is that of "fetch length." This is the distance the wind has travelled before reaching the observer. Wind waves build up over time as the wind blows across the ocean surface, so the longer the fetch length, the larger the waves.

5.1 Choosing a wave spectrum model

You have several choices of a wave model in the `Triton::Ocean::Create()` method. To put it simply - use JONSWAP if you're building a maritime training application, or if you're using Triton to power buoyancy models where ship motion should be realistic. JONSWAP (Joint North Sea Wave Observation Project) is able to use fetch lengths as part of its simulation, and if no fetch lengths are specified, it will automatically fall back to the PIERSON_MOSKOWITZ model. JONSWAP simply extends PIERSON_MOSKOWITZ by simulating the non-linear effects of fetch length, but these effects can be important.

The TESSENDORF, PIERSON_MOSKOWITZ, and JONSWAP models all use inverse Fast-Fourier Transforms to power their wave simulations, so they have the same performance characteristics at runtime. However, they have different computational costs when setting up the FFT's, which happens whenever the simulated wind or swell conditions change. If you anticipate changing the wind direction and/or speed every

frame, you may opt to use TESSENDORF to get the best performance, although this comes at the cost of some physical accuracy (Tessendorf uses a simplified Phillips spectrum, which is very fast to compute.) PIERSON_MOSKOWITZ is slower than Tessendorf, and JONSWAP is the slowest of the three. But again, this only matters if you are changing the simulated conditions every frame.

5.2 Specifying fetch lengths

To take full advantage of the JONSWAP model, you must specify wind fetch lengths as part of your simulation. These are inferred from the WindFetch objects that you pass into Triton via [Triton::Environment::AddWindFetch\(\)](#).

If you call [Triton::WindFetch::SetLocalization\(\)](#) on a wind fetch, Triton may infer the fetch length based on the bounding volume of the wind fetch and the observer's location. The center of the wind fetch represents the origin of the wind, and the fetch length is the distance from the observer to this origin. If the observer is outside the bounds of the WindFetch, its wind is not used.

You may also specify an explicit, constant fetch length using [Triton::WindFetch::SetFetchLength\(\)](#). If set, this overrides use of the bounds defined in [SetLocalization\(\)](#) for computing fetch lengths. However, if [SetLocalization\(\)](#) was called on the WindFetch, its bounds will be used to determine if the WindFetch affects the observer at all.

It is important to use realistic fetch lengths if you are specifying them at all. Typically these are on the order of 100km. The fetch length plays a very important part in the overall wave heights, so setting it too small may result in unnaturally small waves. Setting it too high may yield unnaturally large waves. If you don't have real fetch length data to work with, you're better off leaving it unspecified and using the PIERSON_MOSKOWITZ model instead.

5.3 Simulating swells

Triton may simulate swells from distant storms independently from local wind waves. See the [Triton::Environment::AddSwell\(\)](#) and [Triton::Environment::ClearSwells\(\)](#) methods. These methods give you precise control over the heights, wavelengths, and directions of swell waves that are added into the local wind waves. Typical swell heights are around 3m, and typical swell wavelengths are around 100m - be sure the values you specify are realistic to prevent anomalies. Although you may specify a direction for your swell waves that is different from the local wind direction, this may look unnatural.

Swells are implemented by modifying the input to our inverse FFT to bake in the waves you've specified. As such, you may specify as many swell waves as you would like, and they will have no runtime performance cost.

5.4 Simulating specific Beaufort or Douglas sea states

The users of your application may wish to simulate specific "sea states," and Triton makes this as simple as possible.

The `Triton::Environment::SimulateSeaState()` method will produce local wind waves consistent with the state given from the Beaufort scale. See http://en.wikipedia.org/wiki/Beaufort_scale for a description of the conditions each value describes.

In addition to the sea state simulated, your users may also want to add in specific swell conditions. Just use `Triton::Environment::AddSwell()` to add as many specific swell waves as you wish, and remember to use `Triton::Environment::ClearSwells()` before re-creating the swell conditions.

The Douglas Sea Scale (http://en.wikipedia.org/wiki/Douglas_Sea_Scale) specifies local wind waves and swells independently, so it specifies both kinds of conditions at once. The `Triton::Environment::SetDouglasSeaScale()` method may be used to simulate these conditions. The Douglas sea scale specifies a special "confused" swell state (#9,) which Triton interprets as removing all directionality from the simulated waves.

Chapter 6

Simulating ship wakes with Triton

Triton includes the ability to displace the ocean surface in 3D with wakes generated by multiple ships, or more generally objects moving through the water.

In addition to the standard "Kelvin wake" of 19.46 degrees found behind ships moving at constant velocity in a straight line and a realistically modeled turbulent wake behind the propellers, Triton also properly handles ships that are accelerating, decelerating, or moving along arbitrary paths - all at constant framerates.

Including wakes in your simulation is simple. Just create a [Triton::WakeGenerator](#) object for every ship or object you wish to simulate in your scene, associated with the [Triton::Ocean](#) you'll attach it to.

```
Triton::WakeGeneratorParameters parameters;  
Triton::WakeGenerator *ship = new Triton::WakeGenerator(ocean, parameters);
```

To take advantage of more advanced wake simulation features, you may pass additional information to the WakeGenerator constructor. For example, let's set up a wake for a ship that has a length of 100 meters with particle-based spray effects at the bow (100 meters ahead of the source of the wake), and a beam width of 20 meters. With this information, the turbulent wake behind the ship will expand at a realistic rate given the length and beam width of the ship, which may be useful for training purposes.

```
Triton::WakeGeneratorParameters parameters;  
parameters.sprayEffects = true;  
parameters.bowSprayOffset = 100.0;  
parameters.beamWidth = 20.0;  
parameters.length = 100.0;  
Triton::WakeGenerator *ship = new Triton::WakeGenerator(ocean, parameters);
```

Then, each frame, update the ship propellers' position, direction, and velocity using [Triton::WakeGenerator::Update\(\)](#). The position is in world units, the velocity in world units per second, and the timestamp is in seconds. The direction is a normalized vector in world space.

```
ship->Update(Triton::Vector3(shipX, shipY, shipZ), shipDirection, shipVelocity,  
            now);
```

In C#, it works the same way - just observe the differences in syntax from C++. (Don't use *'s since you don't have pointers, use . instead of ::, etc.)

That's all there is to it! Once you stop calling Update on the WakeGenerator, any existing wakes will dissipate over time. Just delete the WakeGenerator object when you're done with it, and make sure the Triton::Ocean it's attached to is initialized and is not deleted during the WakeGenerator's lifetime. Make sure the velocity is realistic and accurate, in order to receive realistic and accurate wakes.

You may simulate as many WakeGenerators as you'd like in a scene. As the wakes are applied inside Triton's vertex programs, there are a finite amount of individual wake waves that may be drawn in any specific frame. Triton will select the waves closest to the camera if there are more waves being simulated that can be drawn at once. You may adjust this upper limit using the max-wake-waves settings in resources/Triton.config.

Note that the turbulent wake simulation may be taxing on some systems, as it is performed almost entirely on the GPU. If you experience poor performance when wakes are in the scene, try setting the value "wake-propeller-backwash" to "no" inside the file resources/Triton.config, and the propeller backwash behind ships will be disabled. Similarly, the 3D Kelvin wakes may also be disabled using the "wake-kelvin-wakes" setting. Disabling one or the other wake effects will significantly improve performance when wakes are visible, as will adjusting the max-wake-waves parameters for the renderer you are using.

Note also that 3D Kelvin wakes are disabled when using OpenGL on ATI/AMD cards. This is due to what appears to be a driver limitation. If you'd like to try using 3D Kelvin wakes on your AMD card anyhow, set the "disable-kelvin-wakes-AMD" setting in the resources/Triton.config file to "no".

Chapter 7

Spray effects and breaking waves

Triton looks for sharp wave crests and automatically creates particle-based spray effects near the camera.

As the wind increases and the waves become higher, or as the choppiness is increased, the spray effects will become more pronounced.

These particle systems are highly optimized, but still come at a performance cost. If you'd like to disable the spray effects to maximize Triton's performance, just call `Triton::Ocean::EnableSpray()` to turn it on or off at runtime. To disable spray effects entirely, see the `fft-enable-spray` setting in the `resources/Triton.config` file.

Fine control over the appearance of the spray effects is available in `Triton.config`; see the "Spray Settings" section in that file to learn how to adjust the appearance and placement of the spray effects to your liking.

Chapter 8

Rotor wash effects

Triton will also simulate the effects of wind from rotary-wing downwash (or any localized wind source) on the water surface.

The [Triton::RotorWash](#) class makes this easy. Simply instantiate a RotorWash object representing your rotor blades. The constructor takes the diameter of the rotor blades, a flag for enabling spray effects, and a flag indicating if you want to use decal texture effects. Each frame, call [Triton::RotorWash::Update\(\)](#) to update the position and orientation of the rotor blades.

Rotor effects will diminish with distance from the rotor, and are designed to fall off after two rotor diameters. Spray effects on the water surface will be generated within three rotor diameters, and the spray particles will be emitted along the reflection vector from the rotors.

The falloff with distance, radius of the effects, and frequency of particle and wave generation may be configured in the Rotor Wash section of the resources/triton.config file.

```
rotorWash = new Triton::RotorWash(ocean, rotorDiameter, true, true);
rotorWash->Update(rotorPosition, rotorDirection, rotorWindVelocity, currentTime);
```

Take care that the direction passed into [Triton::RotorWash::Update\(\)](#) represents the direction pointing from the water toward the ground - opposite the direction of motion. The velocity is in world units per second, so be sure to convert your velocity units as needed.

The final parameter on the RotorWash constructor indicates whether you want to use decal textures with the rotor wash effect. This can be used to add additional visual detail, with more circular waves and streaks emanating from the rotor wash intersection point. Decal textures come with a moderate performance cost however, so this effect should be used judiciously.

Chapter 9

Simulating impacts on the water

You may also simulate the effects of impacts or explosions on the water, using the [Triton::Impact](#) class.

The size of the waves and spray resulting from the impact are a function of the energy of the impact, which is half of the impactor's mass times the square of its velocity.

For example, you might simulate a bullet of 20 grams hitting the water at 500 m/s with the following code:

```
Triton::Impact *impact = new Triton::Impact(ocean, 0.01, 20.0, true);
DWORD millis = timeGetTime();
impact->Trigger(Triton::Vector3(0, 100, 0), Triton::Vector3(0, -1, 0), 500.0, (double)millis * 0.001);
```

This will produce a small splash, with small ripples radiating from the impact. The 0.01 value in the Impact constructor specifies an object diameter of 1 cm, which will limit the radius of the splash effect.

Or, a torpedo hitting the water has a mass of about 260 kg and a velocity of 100 m/s:

```
Triton::Impact *impact = new Triton::Impact(ocean, 2.0, 260000.0, true);
DWORD millis = timeGetTime();
impact->Trigger(Triton::Vector3(0, 100, 0), Triton::Vector3(0, -1, 0), 100.0, (double)millis * 0.001);
```

This will produce a larger effect. See the documentation for [Triton::Impact::Impact\(\)](#) and [Triton::Impact::Trigger\(\)](#) for more details.

Chapter 10

Applying decals to the water surface

Triton has the ability to apply decal textures to the water surface, which will appear to float on the water.

For example, you could provide Triton with a texture representing an oil film, and apply it over the water at a given location to create an oil slick effect.

Decal textures may be scaled and their blending adjusted at runtime.

Use the [Triton::Ocean::AddDecal\(\)](#) method to add a decal to the water surface. You must pass it a texture to use for the decal, the original size of the decal, and its position on the water surface.

For example:

```
Triton::DecalHandle myDecal = ocean->AddDecal((Triton::TextureHandle) (openGLTextureID), 10.0f, Vector3(20.0, 0, 0));
```

will create a decal using a texture you previously loaded (indicated by `openGLTextureID`), that is 10 meters wide and deep, centered at (20, 0, 0). The `TextureHandle` given to `AddDecal` should be a `GLuint`, `LPDIRECT3DTEXTURE9`, or `ID3D11ShaderResourceView*` depending on whether you are using OpenGL, DirectX9, or DirectX11 respectively.

[Triton::Ocean::ScaleDecal\(\)](#) and [Triton::Ocean::SetDecalAlpha\(\)](#) may be used at runtime to adjust the decal, using the `DecalHandle` returned by `AddDecal()`.

If the decal is no longer needed, use [Triton::Ocean::RemoveDecal\(\)](#) to remove it from the scene. Decals do incur a performance cost, and should be used sparingly.

Chapter 11

Integrating Triton with terrain and shallow water

If your application includes terrain as well as open ocean, there are several ways to achieve good results at the shoreline.

The best approach is to provide Triton with a height map of your terrain, that includes bathymetry data. If a height map is provided using [Triton::Environment::SetHeightMap\(\)](#), the water will automatically become more transparent near the shore, and waves will be dampened to minimize any depth precision artifacts. Take care to only call `SetHeightMap` when your height map changes; Triton needs to copy the texture into system memory, and you don't want to do that every frame. Be sure to create your Ocean object with breaking waves flag enabled to take full advantage of the height map's data.

If you have provided a valid height map, you may also use [Triton::Environment::SetBreakingWavesParameters\(\)](#) to simulate waves breaking at the shoreline. This will produce high-wavelength waves that slow down as they approach the coastline with foam and displacement effects. They look great from the air, but for performance reasons the waves don't actually curl over themselves and generate spray particles when breaking. You'll need to specify the world direction toward the local coastline so the waves travel in the correct direction; generally you'll be fine leaving the other parameters set to their defaults. It's important that your bathymetry data in the height map is realistic; the waves will start to appear as the water becomes more shallow, and realistic depth information will result in waves appearing at the proper distance from shore and following the contour of the coastline.

In addition to a floating-point height map texture, [Triton::Environment::SetHeightMap\(\)](#) also requires a matrix to transform world coordinates into texture coordinates. Here's an example of computing this matrix and creating a height map texture of the proper format using OpenGL; it creates an orthogonal projection with a view looking down at the entire terrain from an altitude of 10000m, in a coordinate system where Y is "up" and the terrain's extents range from (minX, minZ) to (maxX, maxZ).

```
glEnable(GL_TEXTURE_2D);
glGenTextures(1, &heightMap);
glBindTexture(GL_TEXTURE_2D, heightMap);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

```

glTexImage2D(GL_TEXTURE_2D, 0, GL_LUMINANCE32F_ARB, HEIGHT_MAP_DIM, HEIGHT_MAP_DIM, 0, GL_LUMINANCE, GL_FLOAT, (void*)heightData);

glMatrixMode(GL_TEXTURE);
glPushMatrix();
glLoadIdentity();

const GLdouble bias[16] = {
    0.5, 0.0, 0.0, 0.0,
    0.0, 0.5, 0.0, 0.0,
    0.0, 0.0, 0.5, 0.0,
    0.5, 0.5, 0.5, 1.0};
glLoadMatrixd(bias);

double xrange = maxX - minX;
double zrange = maxZ - minZ;

glOrtho(-(xrange * 0.5), xrange * 0.5, -(zrange * 0.5), zrange * 0.5, -1.0, 1.0);

double eyex = minX + xrange * 0.5;
double eyey = 10000.0;
double eyez = minZ + zrange * 0.5;

double centerx = eyex;
double centery = 0;
double centerz = eyez;

double upx = 0;
double upz = -1.0;
double upy = 0;

gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz);

glGetDoublev(GL_TEXTURE_MATRIX, heightMapMatrix);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);

```

You would then pass the resulting height map texture and matrix to Triton like this:

```

tritonEnvironment->SetHeightMap((Triton::TextureHandle)heightMap,
    Triton::Matrix4(heightMapMatrix));

```

DirectX9 users should refer to the documentation for `Triton::Environment::SetHeightMap()` for important information on the type of texture Triton expects.

OpenSceneGraph users may refer to the `OSGDynamicHeightMap` sample included with the SDK. This sample generates a height map from the scene surrounding the camera automatically, and feeds it to Triton for smooth coastline blending. This sample also includes code for a vertex program that can compute accurate height map data for geocentric or ECEF terrains.

If you don't have a height map available for your terrain, be sure to call [Triton::Ocean::SetDepth\(\)](#) if your camera is near the shoreline. This method allows you to specify the depth and surface normal of the seafloor at the camera position. At shallow depths, this information is used to make the waves more pronounced near the shore, and to make the water more transparent near the shoreline. If your terrain includes textured geometry

extending under the water for some distance offshore, your ocean floor will influence the color of the ocean near the shore. The Triton demo application available from our website illustrates this effect when the "Beach" checkbox is enabled.

For handling the water/land boundary without height maps, the simplest approach is to rely on the depth buffer. You'll need to ensure your depth buffer has adequate precision for this to work well. Use a 32 bit depth buffer if your system supports it, and ensure your near and far clip planes are set as close together as is practical. If depth buffer precision issues are a problem in the distance, you may also use the method [Triton::Ocean::SetDepthOffset\(\)](#) to force the water's depth values to be closer to the camera. A value of 0.01 generally seems to clear things up.

If your existing terrain database includes geometry for the ocean surface, you may be better off using [Triton::Ocean::SetPatchShader\(\)](#) to apply Triton's shaders to your existing water geometry. This same technique may be used to render smaller bodies of water, instead of the infinite oceans rendered by [Triton::Ocean::Draw\(\)](#). See [Rendering User-Defined Patches of Geometry](#) for more information.

If your terrain includes areas of land that are below the simulated sea level, you'll need to prevent Triton from rendering waves in these areas. A simple solution is to render your terrain while writing to the stencil buffer, and then enabling the stencil test surrounding the call to [Triton::Ocean::Draw\(\)](#). Or, use [Triton::Ocean::SetPatchShader\(\)](#) with your own water grids to be more selective about where water is rendered in your scene.

You may adjust the sea level of Triton using [Triton::Environment::SetSeaLevel\(\)](#).

Chapter 12

Synchronizing Triton across multiple viewports or channels

In multi-channel simulators that render several viewports at once using multiple computers, you need to ensure that the ocean is rendered consistently across the different systems.

Fortunately, [Triton::Ocean::Draw\(\)](#) takes an explicit time sample as its parameter. As long as you're synchronizing the time passed in across the channels, the display should be consistent.

There is a random component to the wave generation, that just uses the standard `rand()` function. Initialize each channel consistently with `srand()` and you should be fine. Alternately, refer to the documentation for the [Triton::RandomNumberGenerator](#) class - you may subclass this interface, and provide your own random number generator to Triton using the `Environment::SetRandomNumberGenerator()` method.

When running multiple windows on a single computer, you'll need to maintain separate Ocean and Environment instances for each individual graphics context in use. It's important that `Ocean::Draw` is called from the same thread and GL context that the Ocean and Environment were created in.

Chapter 13

Intersection tests with Triton

Triton allows you to query the height and surface normals of the ocean at any given position; this allows you to simulate floating object in your application consistently with Triton's waves.

In order to use intersection tests, you must create your Ocean object with the enable-HeightTests parameter of [Triton::Ocean::Create\(\)](#) set to true. By default, intersection tests are disabled, which allows Triton to avoid the performance hit of reading data back from the GPU on some systems.

With a properly constructed [Triton::Ocean](#), simply call the [Triton::Ocean::GetHeight\(\)](#) method at runtime to query the height and normal of the ocean surface at the intersection of a given ray and the ocean surface. For example:

```
Triton::Vector3 testPos(0.0, 100.0, 0.0);
Triton::Vector3 down(0.0, -1.0, 0.0);
Triton::Vector3 normal;
float height = 0;
if (ocean && ocean->GetHeight(testPos, down, height, normal, true)) {
    // do something with the height at this point...
}
```

The height returned will be accurate to within one grid vertex of the ocean mesh.

Chapter 14

Underwater rendering with Triton

Triton provides support for rendering the water surface from above or below sea level.

Rendering the rest of the scene's objects underwater is the responsibility of the application, but we provide hooks to ensure the sea surface is rendered consistently with the rest of your scene.

You'll want to take a look at the method [Triton::Environment::SetBelowWaterVisibility\(\)](#). In addition to rendering the water surface properly from below the water, this method will allow you to fog the water surface to a given visibility and fog color.

When underwater, setting a fog color that's consistent with the background and fog used for the rest of the underwater scene will yield good results. Clearing the back-buffer to match the color specified, or using a skybox with a specific color below the horizon, will work well.

Note that these methods take in a visibility value; this will be translated into an exponential fog extinction value using the Koschmieder equation: $\text{visibility} = 3.912 / \text{extinction}$.

Triton also has an underwater "God Rays" effect that may be triggered with the [Triton::Ocean::EnableGodRays\(\)](#) method. When enabled, shafts of light will automatically be drawn while underwater and near the surface, looking toward the refracted sunlight vector. You may adjust the look of this effect with the God Rays section of the `resources/Triton.config` file. These shafts of light are generated from the wave motion at the water surface, so the animation of this effect will be more intense in rough seas, and less intense in calm seas.

Chapter 15

Using Triton with No Rendering (Physics Only)

Some systems consist of a central server that is responsible for intersection tests and physics, that does not do any rendering.

Triton may be integrated into such systems without requiring communication to visualization channels.

All you need to do is specify `NO_RENDERER` for the "renderer" parameter of [Triton::Environment::Initialize\(\)](#). In this mode, Triton will not do any drawing, but will still compute the physics of the water surface, complete with GPU acceleration if available.

Follow the advice in [Synchronizing Triton across multiple viewports or channels](#) to ensure the physics channel is in sync with the rendering channels. You may then use [Triton::Ocean::GetHeight\(\)](#) to obtain intersections and normals with the water surface, as described in [Intersection tests with Triton](#)

Chapter 16

Performance tuning tips

Triton will use as many special capabilities as it can find of your graphics hardware, taking advantage of CUDA, OpenCL, and the parallel computing capabilities of your main CPU whenever possible.

If you need even more performance, there are some things you can tweak.

If your engine or framework performs updates in a different pass and/or thread from rendering, have a look at the `Ocean::UpdateSimulation()` method. You may use this to perform the FFT calculations for Triton separately from the actual drawing in `Ocean::Draw()`, which can help performance in some situations.

If you believe vertex processing may be your bottleneck, you can decrease the number of polygons used in the projected grid used to render the ocean. Open up the file `resources/Triton.config` in a text editor, and reduce the value of `default-grid-resolution`. As the number of polygons increases with the square of this value, reducing it can have a big impact on some systems.

Computing the Fast Fourier Transforms associated with the TESSENDORF wave model can also be expensive. You can reduce the cost of these computations by reducing the values of `fft-grid-dimension-x` and `fft-grid-dimension-y` to a smaller power of 2. You may want to also reduce the `fft-grid-size` settings to match in order to avoid a loss in resolution (at the cost of increased tiling.)

If you're feeling adventurous, you can also edit the shaders used by Triton to simplify them. You'll find them in the `resources` folder; eliminating the sections that apply foam and noise may help performance slightly, but in our experience the small cost of these effects is well worth the visual quality.

Chapter 17

Troubleshooting tips

17.1 I see aliasing or shimmering on the ocean.

Since Triton draws its waves using a fine-grained screen-aligned grid, it's vulnerable to aliasing artifacts if your driver's quality settings are set too low. Make sure you've got anti-aliasing enabled in your application and your drivers have good texture filtering options enabled.

17.2 I can't link the Triton libraries

Ensure you have all of the latest service packs for Visual Studio installed on your system. Also, check for preprocessor defines in your application that affect STL linkage, such as `_SECURE_SCL` or `_HAS_ITERATOR_DEBUGGING`. If problems persist, contact support@sundog-soft.com; we can provide you with obfuscated source code to allow you to build the Triton libraries with whatever development environment and compile time flags you need. If you are building your project with a statically linked runtime library (ie, "Multithreaded" vs. "Multithreaded DLL"), you'll probably need to add `MSVCRT` to the list of runtime libraries to ignore in your linker settings. Make sure the Triton library you are linking against matches the runtime you are using. For example, "Multithreaded Debug DLL" should use `Triton-MTD-DLL.lib`.

17.3 My Ocean is black!

Make sure you're calling `Ocean::SetAmbientLight()` and `Ocean::SetDirectionalLight()` prior to calling `Ocean::Draw()`.

17.4 My water is all sorts of strange colors!

Double check the direction being passed in to [Triton::Environment::SetDirectionalLight\(\)](#). If you're inadvertently passing in the direction from the light source instead of the direction to the light source, the color of the water can be undefined. Try negating what you're passing in there.

17.5 Nothing's showing up at all!

Make sure you're checking for errors returned from [Triton::Environment::Initialize\(\)](#). If this method fails, you'll get an informative error code back - which will most likely tell you that the path to Triton's resources directory specified in the constructor of [Triton::ResourceLoader](#) was invalid.

Also ensure you're getting back a valid [Triton::Ocean](#) object from [Triton::Ocean::Create\(\)](#). This method may return NULL, which likely indicates a driver compatibility issue we haven't encountered before. The first thing to try is updating your graphics drivers to the latest release, but if that fails, we provide a way to get more information about what's going on with Triton under the hood.

Open up the file resources/Triton.config in a text editor, and change the setting enable-debug-messages to "yes". Now, when you run your application in Debug mode, you'll get information in the Output window of Visual Studio prefaced with "TRITON:" that tells you more about how it selected its underlying FFT method, and error information.

If these messages implicate a specific FFT implementation as running into problems, the simplest thing is to disable the culprit. The settings disable-cuda, disable-ipp, disable-opencl, and disable-compute-shader will let you force Triton to not use a FFT method that's potentially problematic on your system. You may want to try enabling fft-force-cpu to force Triton to use its built-in CPU-based FFT transform, which has no special system dependencies or DLL dependencies at all.

If you receive the message "Failed to initialize projected grid" after calling [Triton::Ocean::Create\(\)](#) with no other messages before it, you may be creating the Ocean from a different thread than you used to create the GL context. Make sure all of your Triton calls are done in the same thread as the one your context was created within.

Another likely culprit is the matrices passed into Triton via [Triton::Environment::SetCameraMatrix\(\)](#) and [Triton::Environment::SetProjectionMatrix\(\)](#). Double check that these matrices contain what you expect, and try transposing them in case your engine's conventions differ from ours. If you just can't get the transforms right, try creating these matrices "from scratch" from your camera properties, as illustrated in the sample code provided with the SDK.

17.6 My ocean looks garbled or corrupt.

There are some known issues with Intel integrated graphics and how writes to floating point textures are handled, which can lead to missing or garbled wave heights. Try updating your drivers.

Some older ATI drivers have problems with mipmap generation in OpenGL. If you see black squares in the ocean that get worse with distance, the mipmaps on our textures are likely failing. Updating your drivers should clear this up.

17.7 Triton crashed on me

If Triton crashes inside `Ocean::Draw()`, ensure that the `Draw()` call is happening from the same thread and GL context that the `Ocean` and `Environment` were created in. Common causes of this issue are initializing Triton before the graphics context is initialized, or multiple window setups where you'll need to maintain individual `Environment` and `Ocean` instances per window (unless you have enabled context list sharing between these multiple graphics contexts.)

If your application was built with the preprocessor flags `_SECURE_SCL=0` or `_HAS_ITERATOR_DEBUGGING=0`, this can also lead to trouble since our libraries are not built with those flags. Try removing those flags, or contact support@sundog-soft.com to get a special library build for your needs.

If you're still running into trouble, or you believe you've encountered a system compatibility issue we should know about, please send us a note at support@sundog-soft.com. We're happy to provide limited pre-sales support to you, and we're always very interested in identifying and fixing any new bugs we haven't come across before.

Chapter 18

Redistributing Triton with your application

Windows developers must ensure that the DirectX end-user runtimes are installed on your target systems, from the June 2010 DirectX SDK or newer.

If you are developing with Visual Studio 2012 and using DirectX, you will also need to install the run-time shader compiler DLL's alongside your application's executable file. You'll find these files as `d3dcompiler_46.dll` and `d3dcsx_46.dll` inside the `resources/dll` and `resources/dll64` directories; choose the DLL's for the architecture you're building for. Visual Studio 2013 users will need to do the same, but using the `_47` versions of these files.

If you have a full source license and are targeting OpenGL exclusively, it is possible to compile Triton such that all DirectX dependencies are removed. If you define `DIRECTX9_FOUND=0` and `DIRECTX11_FOUND=0` in your preprocessor settings, you may then remove the DirectX libraries from the project safely.

Triton's runtime dependencies are contained within the "resources" directory of the SDK, which you are free to redistribute with your application. You're also free to roll the contents of this directory into your own resource manager if you wish; see [Integrating with your own resource manager](#) for more information. Linux users must also ensure that the IPP, `clAmdFft`, and `cufft` shared objects that we installed into your `/usr/local/lib/triton` directory are installed on your target systems in locations that are part of the library search path.

If Windows developers want to trim down the size of the resources directory, it's safe to remove any of the DLL's in the `resources/vcX` directories that you're not using. If you built your solution with Visual Studio 2008, it's safe to delete the entire `vc8`, `vc10`, and `vc11` directories. If your application's built for `win32` instead of `x64`, the `x64` subdirectory can go. Then, within your surviving directory, any DLL's for runtime libraries you aren't using are safe to remove as well. You'll also find two directories containing runtime dependencies for triton: `resources/dll` and `resources/dll64`. If your application is built for `x64`, it's safe to remove the `dll` folder. If it's built for `Win32`, it's safe to remove `dll64`.

You may also remove shaders from the resources directory that you aren't using. If your application is for DirectX only, all the .glsl files can go, for example - or, OpenGL users may safely remove the .fx files.

As with any Windows applications, if you link against the DLL runtime libraries, you'll also need to ensure that the Visual C++ runtimes for your compiler and architecture are installed on your target systems as well. Make sure you install the latest available service pack runtimes for the version of Visual Studio you're using, and take care to install the 32 or 64 bit versions of them as appropriate.

If you are using Visual Studio 2013, you will need to install both the latest Visual Studio 2013 and Visual Studio 2012 runtime libraries on the end users' system, if you are linking against DLL runtimes. The version of CUDA we use does not support Visual Studio 2013, and so it depends on the Visual Studio 2012 toolchain instead.

Chapter 19

Obtaining support

We're happy to provide limited email-based pre-sales technical support if you're evaluating Triton, and licensed Pro customers receive 3 months of support as well.

Just contact support@sundog-soft.com with your questions. Be sure to include which renderer you're using, what your graphics card and driver version is, and what kind of CPU you're using so we can better help you.

Chapter 20

Advanced topics

20.1 Integrating with your own resource manager

It's possible to derive your own [Triton::ResourceLoader](#) class to hook into your own resource manager. Implement your own [Triton::ResourceLoader::LoadResource\(\)](#) and [Triton::ResourceLoader::FreeResource\(\)](#) methods to grab Triton's shader and graphical resources, located in the resources directory of the SDK, from any resource database you might have. Then, pass your derived [Triton::ResourceLoader](#) into [Triton::Environment::Initialize\(\)](#) and it will be called instead of our default one.

The DLL's within our resources directory will require special care, however. You'll need to keep the vc9 and/or vc10 subdirectories of the resources directory in place on disk, and continue to pass a valid path to the resources directory to the [Triton::ResourceLoader](#) constructor so Triton will know where to find our FFT DLL's. Failure to do so will force Triton to fall back on a CPU-only FFT transform, which will hurt performance in a big way.

You'll also need to ensure the third-party DLL's located in the resources/dll directory are distributed in a place where Triton will be able to load them as part of the DLL search path. Our default implementation of [Triton::ResourceLoader::SetResourceDirPath\(\)](#) calls the Windows function `SetDllDirectory` to add this directory to the DLL search path. You'll want to emulate this behavior, or redistribute these DLL's alongside your application's executable or working directory so Triton will find them when loaded.

20.2 Integrating with your own memory manager

If you'd like to hook into your own memory manager instead of using ours (which is just based on the Windows functions `HeapAlloc()` and `HeapFree()`), it's possible to do so.

Extend your own [Triton::Allocator](#) class from the one defined in [MemAlloc.h](#). Then, pass it in via the static [Triton::Allocator::SetAllocator\(\)](#) method prior to calling any other Triton methods or instantiating any Triton objects.

Since Triton operates across DLL boundaries, you'll want to take care that a consistent heap is used, as we do. Licensed users have access to Triton's full source, and you may find it informative to examine our own implementation of [Triton::Allocator](#) before creating your own.

We take care to capture every usage of new, delete, malloc, and free within Triton, as well as hooking into any memory allocated from STL objects. Memory may be allocated by system functions and third party libraries (such as CUDA, OpenCL, and FFTSS) that is outside of our control, however.

20.3 Adding your own effects to the water

You'll find Triton's shaders in plain text inside the Resources folder; .glsl files are for OpenGL and .fx are effect files for DirectX9 and DirectX11. You're free to make changes to these shaders as you see fit to customize your application, or integrate more tightly with your graphics engine. For example, you could introduce support for additional light sources.

If you are developing changes to Triton's shaders, it's a good idea to enable the enable-debug-messages setting in resources/Triton.config so you'll see any compilation errors in the shaders. Any error messages will appear in your debugger's output log.

Your modifications will likely require additional parameters to be passed in to the shaders. To enable this, we offer the [Triton::Ocean::GetShaderObject\(\)](#) method. Depending on the renderer being used, this will return to you a GLhandle, a ID3DXEffect pointer, or a ID3DX11Effect pointer, which you can use to pass in uniform parameters of your own - for example, your own textures or matrices used to transform them as they are combined with the water. GetShaderObject() takes a parameter indicated which shader you wish to modify. For the ocean surface, be sure to apply your uniforms to both the WATER_SURFACE and WATER_SURFACE_PATCH programs. These shader programs draw the water when using a projected grid and a conventional mesh, respectively.

If you'd like to keep your modified shaders separate from Triton's stock shaders, you can use the config setting "shader-filename-prefix" in resources/Triton.config to add a prefix to the shader file names when they are loaded. For example, setting "shader-filename-prefix = my-" would result in the file "my-flat-fft.fx" being used instead of "flat-fft.fx".

We also offer the "user-functions.glsl" file, which offers easy to use hooks into the lighting of all of Triton's geometry. Refer to the comments in this file to learn how to override Triton's shading. By placing your custom shading code into user-functions.glsl, you can avoid modifying our shaders directly, which makes updating Triton easier. This also has the benefit of placing your custom shading code in one place, instead of worrying about extending all of Triton's different individual shader programs for geocentric, flat, projected grid, and conventional mesh rendering.

For deeper integration, we offer full source licenses in addition to binary licenses on our website (www.sundog-soft.com). However, even with a binary license you receive the shader sources and the ability to hook into them as described here.

20.4 Building Triton from source

Licensed customers of the full source Triton Ocean SDK will receive a special SDK installer that includes Triton's full source code, enabling you to modify Triton to meet your own special needs.

You'll find solution files for Visual Studio 2005, 2008, 2010, and 2012 included at the top level directory of the SDK.

In order to compile the TritonOpenCL DLL project, you'll need to have AMD's AMD Accelerated Parallel Processing (APP) SDK installed.

In order to compile the TritonCUDA DLL project, you'll need NVidia's CUDA Toolkit 6.0 or newer installed.

Both are freely available from AMD's and Nvidia's websites.

You'll find that the documentation included with the full source SDK includes references on all of Triton's internal classes, in addition to the public API's.

If you have any trouble building Triton, drop a note to support@sundog-soft.com.

20.5 Integrating planar reflection maps with Triton

In addition to environment cube maps, you may also pass in planar reflection maps for use with Triton. This is useful for generating local reflections from ships and terrain. Use `Triton::Environment::SetPlanarReflectionMap()` for this effect.

The type of texture parameter passed into this method will vary depending on what renderer you're using. OpenGL users should pass in a `GLuint` representing the texture ID of the planar reflection texture. DirectX9 users should pass in a `LPDIRECT3DTEXTURE2D`. DirectX11 users should pass a `ID3D11ShaderResourceView` pointer.

Triton can use planar reflection and environment map textures together. If both are applied, the alpha channel of the planar reflection texture is used to blend between planar reflection and environment cube map reflections. This is a good way to get the "best of both worlds," with the cube map providing environmental reflections from steep wave angles, and the planar reflection map providing local reflections directly above the water surface.

Generation of planar reflection maps can be considered to be an advanced topic. It requires a render to texture pass to produce a proper reflection map. The scene rendered to such a texture has to be mirrored in the reflection plane by scaling the height values by -1 (this can be accomplished by multiplying a scaling matrix into the reflection camera's view matrix). Such scaling flips winding directions, so will be necessary to change the polygon winding order used for backface culling when rendering this reflection camera. Application of user clip planes to cut off pieces of scene models normally hidden below the water surface may be also required. All these topics are described in great detail in various documents available on the Internet. If you seek more info try searching the web for "Water Rendering" and you will surely find plenty of documentation on the topic of rendering mirror textures.

Together with a planar reflection texture, Triton requires passing a texture matrix used

to project Triton's computed reflection vector to texture coords. In the most common scenario, this is exactly the View*Projection matrix used to render the main camera, scaled and translated to 0..1 x 0..1 texture coord space. Usually, you construct the texture matrix for the reflection map lookup by first zeroing out any translation in your main scene's view matrix, then multiplying by the projection matrix, then multiplying by a transformation matrix that translates by (1, 1, 1) and scales by (.5, .5, .5) to get to texture coordinates. (DirectX users will want to translate by (1, 1, 0) and scale by (0.5, -0.5, 1.0) due to different conventions.) However, more complex scenarios aiming to optimize the use of reflection map space can adopt less intuitive View and Projection matrices. Such unusual TextureMatrices can be also passed to [Triton::Environment::SetEnvironmentMap\(\)](#).

For example, here's some OpenSceneGraph code that computes the proper texture matrix to pass in for [Environment::SetPlanarReflectionMap](#), assuming the texture was rendered from a camera that flipped the height coordinates using the same view and position as the main camera:

```
view.setTrans( 0, 0, 0 );
_textureProjectionMatrix->set( view *
                               projection *
                               osg::Matrix::translate( 1,1,1 ) *
                               osg::Matrix::scale( 0.5, 0.5, 0.5 ) );
```

Whatever matrix you pass in will be applied to the view vector in our fragment shaders, prior to projective texture mapping to the reflection texture. Our shaders are all located inside the Resources folder and compiled at runtime, so if you need to modify them to support alternative reflection schemes you are free to do so.

Triton does provide a helper function to compute commonly used reflection and texture matrices for planar reflections with [Triton::Ocean::ComputeReflectionMatrices\(\)](#). This method will give you a 4x4 matrix to flip your scene about the local water surface, and a 3x3 matrix that can be used for texture lookups into your reflection texture. Here's some pseudo-code in OpenGL illustrating how to use it to create a valid reflection texture map and matrix suitable for [Environment::SetPlanarReflectionMap\(\)](#).

```
// The following code renders models into a texture as part of the reflection map
pass.

glMatrixMode(GL_MODELVIEW);
glPushMatrix();

// Apply the camera position and rotation.
glLoadIdentity();
glRotated(pitch, 1, 0, 0);
glRotated(yaw, 0, 1, 0);
glTranslated(-camX, -camY, -camZ);

// Pass the camera info to Triton
double mv[16], proj[16];
glGetDoublev(GL_MODELVIEW_MATRIX, mv);
glGetDoublev(GL_PROJECTION_MATRIX, proj);
tritonEnvironment->SetCameraMatrix(mv);
tritonEnvironment->SetProjectionMatrix(proj);

// Ask Triton for reasonable matrices to use
Triton::Matrix3 reflectTexMatrix;
```

```

Triton::Matrix4 reflectionMatrix;

if (tritonOcean->ComputeReflectionMatrices(reflectionMatrix, reflectTexMatrix))
{
    // Flip everything about the water surface
    glMultMatrixf(reflectionMatrix.ToFloatArray());

    // Activate a render to texture target for your reflection texture
    // Setting up and using an FBO is a lot of code so we're not showing
    // that here, contact support@sundog-soft.com if you need help.
    framebufferObject->SetupRenderPass();

    // Flip the back face culling winding order, since we flipped everything
    glFrontFace(GL_CW);

    // Set up a user clipping plane to prevent reflection geometry under water
    // There are also tricks for doing this by manipulating the near clip plane
    // of your projection matrix if you want to look that up.
    double planeEq[4] = {0, 1, 0, tritonEnvironment->GetSeaLevel()};
    glClipPlane(GL_CLIP_PLANE0, planeEq);
    glEnable(GL_CLIP_PLANE0);

    // Draw anything you want reflected in the water. Note this is additive to
    // any environmental cube map reflections you may have, so you can handle
    // sky reflections separately and less frequently.
    DrawModels(true);

    glDisable(GL_CLIP_PLANE0);

    // Close out your render texture and restore things the way they were
    framebufferObject->EndRenderPass();

    glFrontFace(GL_CCW);

    // Give it to Triton
    tritonEnvironment->SetPlanarReflectionMap((Triton::TextureHandle) (frameBuffer
        Object->GetTexture()),
        reflectTexMatrix);

    // Blend the reflection with other reflections 60%.
    tritonOcean->SetPlanarReflectionBlend(0.6f);
}

glPopMatrix();

```

Note that we also used [Triton::Ocean::SetPlanarReflectionBlend\(\)](#) to control the strength of the planar reflections in the final output. You might use this to have stronger reflections in calm water, or to fade out reflections entirely as wave heights increase or prior to disabling your reflection pass.

20.6 Restricting Use of Computing Resources

Triton achieves its performance by using any parallel computing resources it can find, including your GPU's and multi-core CPU's. By default, it will typically max out usage of your GPU and CPU in order to render the water as fast as possible. In practice, this means more time is left each frame to render your other objects in the scene.

Some users find this behavior undesirable due to the power usage and heat generated

from running your computer so hard, or perhaps it interferes with scheduling of other parallel processes in your application. We allow you to control how aggressively Triton uses your computing resources to mitigate these concerns.

If you need to preserve CPU resources, `Triton::Environment::EnableOpenMP()` may be used to prevent use of OpenMP to split up intensive loops across multiple cores. This will restrict Triton to using the core it was invoked on, unless Triton is falling back to CPU-based FFT computations.

If you need to preserve GPU resources, you may want to disable Triton's use of CUDA, OpenCL, and DirectX11 Compute Shaders to force all FFT computations to happen on the CPU instead. The config settings `disable-cuda`, `disable-opencl`, and `disable-compute-shader` in `resources/triton.config` may be used to achieve this.

20.7 Using linear color space with Triton

If your application renders its scene internally in linear color space, you'll need to tell Triton to correct its shaders to remove the effects of gamma correction as well.

The method `Triton::Ocean::SetLinearColorSpace()` may be used for this purpose. If set to true, the ocean surface colors will be raised to the power of 2.2 (which un-does gamma correction of 2.2.) If false, the ocean surface colors are left unchanged.

Chapter 21

Third-party license notices

Triton incorporates some third-party code, and their required license notices are here.

Triton's own license terms are found in the file `license.txt` installed with your SDK, and were presented to you upon installation. We do not include any GPL software in Triton, and there are no demands to make your product "open source" in any of these terms.

All third party trademarks and logos are the property of their respective owners.

21.1 FFTSS: A Fast Fourier Transform Library

Copyright 2002-2007 Akira Nukada. All rights reserved. Copyright 2002-2007 The Scalable Software Infrastructure Project, supported by "Development of Software Infrastructure for Large Scale Scientific Simulation" Team, CREST, JST. Akira Nishida, Department of Computer Science, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE SCALABLE SOFTWARE INFRASTRUCTURE PROJECT "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE SCALABLE SOFTWARE INFRASTRUCTURE PROJECT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED

TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

21.2 Intel's Integrated Performance Primitives

Decompiling or reverse engineering the Intel Integrated Performance Primitives DLL's (ipp*.dll) included in the Triton SDK's resources/dll directory is expressly prohibited.

21.3 Cgtextures.com

The sky box textures used in our sample applications are courtesy of cgtextures.com, and may not be redistributed.

21.4 AMD Accelerated Parallel Processing Math Libraries

END USER LICENSE AGREEMENT PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THESE TERMS AND CONDITIONS, DO NOT USE THE SOFTWARE.

1. License. The software accompanying this License (hereinafter "Software", regardless of the media on which it is distributed, are licensed to you by Advanced Micro Devices, Inc. ("AMD"). You own the medium on which the Software is recorded, but AMD and AMD's Licensors (referred to collectively as "AMD") retain title to the Software and related documentation. You may: a) use the Software.; and b) make a reasonable number of copies necessary for the purposes of this License. You must reproduce on such copy AMD's copyright notice and any other proprietary legends that were on the original copy of the Software

2. Restrictions. The Software contains copyrighted and patented material, trade secrets and other proprietary material. In order to protect them, and except as permitted by applicable legislation, you may not: a) decompile, reverse engineer, disassemble or otherwise reduce the Software to a human-perceivable form; b) modify, network, rent, lend, loan, distribute or create derivative works based upon the Software in whole or in part; or c) electronically transmit the Software from one computer to another or over a network or otherwise transfer the Software except as permitted by this License.

3. Termination. This License is effective until terminated. You may terminate this License at any time by destroying the Software, related documentation and all copies thereof. This License will terminate immediately without notice from AMD if you fail to comply with any provision of this License. Upon termination you must destroy the Software, related documentation and all copies thereof.

4. Government End Users. If you are acquiring the Software on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees the Software and documentation were developed at private expense and are provided with "RESTRICTED RIGHTS". Use, duplication, or disclosure by the Government is subject to restrictions as set forth in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1) (ii) (Oct 1988), FAR 12.212(a)(1995), FAR 52.227-19, (June 1987) or FAR 52.227-14(ALT III) (June 1987), as amended from time to time. In the event that this License, or any part thereof, is deemed inconsistent with the minimum rights identified in the Restricted Rights provisions, the minimum rights shall prevail.

5. No Other License. No rights or licenses are granted by AMD under this License, expressly or by implication, with respect to any proprietary information or patent, copyright, trade secret or other intellectual property right owned or controlled by AMD, except as expressly provided in this License.

6. Additional Licenses. DISTRIBUTION OR USE OF THE SOFTWARE WITH AN OPERATING SYSTEM MAY REQUIRE ADDITIONAL LICENSES FROM THE OPERATING SYSTEM VENDOR. Additional third party licenses may also be required and you agree that you shall be solely responsible for obtaining such license rights.

7. Disclaimer of Warranty on Software. You expressly acknowledge and agree that use of the Software is at your sole risk. The Software and related documentation are provided "AS IS" and without warranty of any kind and AMD EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, ACCURACY, CONDITION, OWNERSHIP, FITNESS FOR A PARTICULAR PURPOSE, AND/OR OF NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS, AND THOSE ARISING FROM CUSTOM OR TRADE OR COURSE OF USAGE. AMD DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. FURTHERMORE, AMD DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY AMD OR AMD'S AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT AMD OR AMD'S AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. THE SOFTWARE IS NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING OR LIFE SUSTAINING APPLICATIONS. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

8. Limitation of Liability. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL AMD, OR ITS DIRECTORS, OFFICERS, EMPLOYEES OR AGENTS ("AUTHORIZED REPRESENTATIVES"), BE LIABLE TO YOU FOR ANY

PUNITIVE, EXEMPLARY, DIRECT, INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE, MISUSE OR INABILITY TO USE THE SOFTWARE OR RELATED DOCUMENTATION, BREACH OR DEFAULT, INCLUDING THOSE ARISING FROM INFRINGEMENT OR ALLEGED INFRINGEMENT OF ANY PATENT, TRADEMARK, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT, BY AMD, EVEN IF AMD OR AMD'S AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. AMD will not be liable for: 1) loss of, or damage to, your records or data; or 2) any damages claimed by you based on any third party claim. In no event shall AMD's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Software.

9. Export Restrictions. You shall adhere to all U.S. and other applicable export laws, including but not limited to the U.S. Export Administration Regulations (EAR), currently found at 15 C.F.R. Sections 730 through 744. Further, pursuant to 15 C.F.R. Section 740.6, You hereby certifies that, except pursuant to a license granted by the United States Department of Commerce Bureau of Industry and Security or as otherwise permitted pursuant to a License Exception under the U.S. Export Administration Regulations ("EAR"), You will not (1) export, re-export or release to a national of a country in Country Groups D:1 or E:2 any restricted technology, software, or source code it receives from AMD, or (2) export to Country Groups D:1 or E:2 the direct product of such technology or software, if such foreign produced direct product is subject to national security controls as identified on the Commerce Control List (currently found in Supplement 1 to Part 774 of EAR). For the most current Country Group listings, or for additional information about the EAR or Recipient's obligations under those regulations, please refer to the U.S. Bureau of Industry and Security's website at <http://www.bis.doc.gov/>. These export requirements shall survive any expiration or termination of this Agreement.

10. Controlling Law and Severability. This Agreement will be governed by and construed under the laws of the State of California without reference to its conflicts of law principles. The rights and obligations under this Agreement shall not be governed by the United Nations Convention on Contracts or the International Sale of Goods, the application of which is expressly excluded. Each party hereto submits to the jurisdiction of the state and federal courts of Santa Clara County and the Northern District of California for the purpose of all legal proceedings arising out of or relating to this Agreement or the subject matter hereof. Each party waives any objection which it may have to contest such forum.

11. Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the Software and the related documentation, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of AMD.

Chapter 22

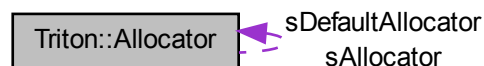
Class Documentation

22.1 Triton::Allocator Class Reference

You may extend the [Allocator](#) class to hook your own memory management scheme into Triton.

```
#include <MemAlloc.h>
```

Collaboration diagram for Triton::Allocator:



Public Member Functions

- virtual void *TRITONAPI [alloc](#) (size_t bytes)
Allocate a block of memory; defaults to malloc()
- virtual void TRITONAPI [dealloc](#) (void *p)
Free a block of memory; defaults to free()

Static Public Member Functions

- static [Allocator](#) *TRITONAPI [GetAllocator](#) ()

Retrieves the static allocator object.

- static void TRITONAPI [SetAllocator](#) ([Allocator](#) *a)

Sets a new static allocator object.

22.1.1 Detailed Description

You may extend the [Allocator](#) class to hook your own memory management scheme into Triton. Instantiate your own implementation of [Allocator](#), and pass it into [Allocator::SetAllocator](#) prior to calling any other Triton methods or instantiating any Triton objects.

Each object in Triton overloads the new and delete operators, and routes memory management through the [Allocator](#) as well.

22.1.2 Member Function Documentation

22.1.2.1 static void TRITONAPI Triton::Allocator::SetAllocator ([Allocator](#) * a) [inline, static]

Sets a new static allocator object.

If this is not called, the default implementation using malloc and free is used.

The documentation for this class was generated from the following file:

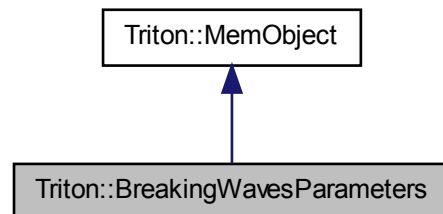
- C:/triton/trunk/Public Headers/[MemAlloc.h](#)

22.2 Triton::BreakingWavesParameters Class Reference

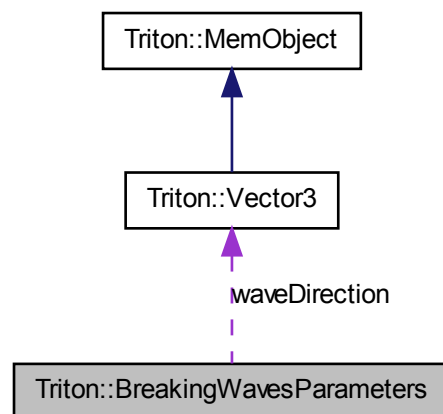
Parameters to control behavior of breaking waves at shorelines, used by Environment::SetBreakingWaves().

```
#include <Environment.h>
```

Inheritance diagram for Triton::BreakingWavesParameters:



Collaboration diagram for Triton::BreakingWavesParameters:



Public Member Functions

- [BreakingWavesParameters](#) ()

The constructor sets reasonable default values, except for the waveDirection member which we can't really guess at.

- void [SetSteepness](#) (float pSteepness)

The "k" value controlling the steepness of the waves; 0 is rounded sine wave, 1.0 is pointy.

- void [SetWavelength](#) (float pWavelength)
The starting wavelength of the breaking waves.
- void [SetWavelengthVariance](#) (float pWavelengthVariance)
How much the wavelength varies as the wave approaches the shore.
- void [SetWaveDirection](#) (const [Vector3](#) &pDirection)
The normalized direction vector pointing toward the shoreline.
- void [SetAutoWaveDirection](#) (bool on)
Sets whether the direction of the waves should be automatically determined by examining the overall slope of the terrain described by the current height map.
- void [SetAmplitude](#) (float pAmplitude)
The amplitude of the breaking waves.
- void [SetSurgeDepth](#) (float pSurgeDepth)
The depth at which the wavelength will rapidly expand to simulate surging surf.
- void [SetSteepnessVariance](#) (float pSteepnessVariance)
The variance in steepness as the wave approaches the shore.
- void [SetDepthFalloff](#) (float pFalloff)
How quickly breaking waves fade off as a function of water depth.

22.2.1 Detailed Description

Parameters to control behavior of breaking waves at shorelines, used by `Environment::SetBreakingWaves()`.

22.2.2 Constructor & Destructor Documentation

22.2.2.1 `Triton::BreakingWavesParameters::BreakingWavesParameters ( )`

The constructor sets reasonable default values, except for the `waveDirection` member which we can't really guess at.

22.2.3 Member Function Documentation

22.2.3.1 `void Triton::BreakingWavesParameters::SetAmplitude ( float pAmplitude )` [inline]

The amplitude of the breaking waves.

Default: 3.0

22.2.3.2 void Triton::BreakingWavesParameters::SetAutoWaveDirection (bool *on*)
[inline]

Sets whether the direction of the waves should be automatically determined by examining the overall slope of the terrain described by the current height map.

If false, the explicit wave direction set via [SetWaveDirection\(\)](#) will be used instead. Defaults to off.

22.2.3.3 void Triton::BreakingWavesParameters::SetDepthFalloff (float *pFalloff*)
[inline]

How quickly breaking waves fade off as a function of water depth.

1.0 will give you "physically realistic" results, but often not what you expect. Larger values will cause breaking waves to fade away closer to shore. Default: 5.0

22.2.3.4 void Triton::BreakingWavesParameters::SetSteepness (float *pSteepness*)
[inline]

The "k" value controlling the steepness of the waves; 0 is rounded sine wave, 1.0 is pointy.

This will automatically increase as the wave approaches the shore; you're just specifying the starting value here. Default: 0.5

22.2.3.5 void Triton::BreakingWavesParameters::SetSteepnessVariance (float *pSteepnessVariance*) [inline]

The variance in steepness as the wave approaches the shore.

Default: 0.5

22.2.3.6 void Triton::BreakingWavesParameters::SetSurgeDepth (float *pSurgeDepth*)
[inline]

The depth at which the wavelength will rapidly expand to simulate surging surf.

Default: 8.0

22.2.3.7 void Triton::BreakingWavesParameters::SetWaveDirection (const Vector3 & *pDirection*) [inline]

The normalized direction vector pointing toward the shoreline.

Used only if SetAutoWaveDirection(false) has been called, or if automatic detection of the ocean floor's slope fails for some reason.

22.2.3.8 void Triton::BreakingWavesParameters::SetWavelength (float *pWavelength*)
[inline]

The starting wavelength of the breaking waves.

Default: 1500.0

22.2.3.9 void Triton::BreakingWavesParameters::SetWavelengthVariance (float
pWavelengthVariance) [inline]

How much the wavelength varies as the wave approaches the shore.

Default: 500.0

The documentation for this class was generated from the following file:

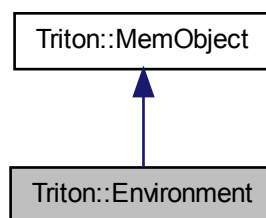
- C:/triton/trunk/Public Headers/[Environment.h](#)

22.3 Triton::Environment Class Reference

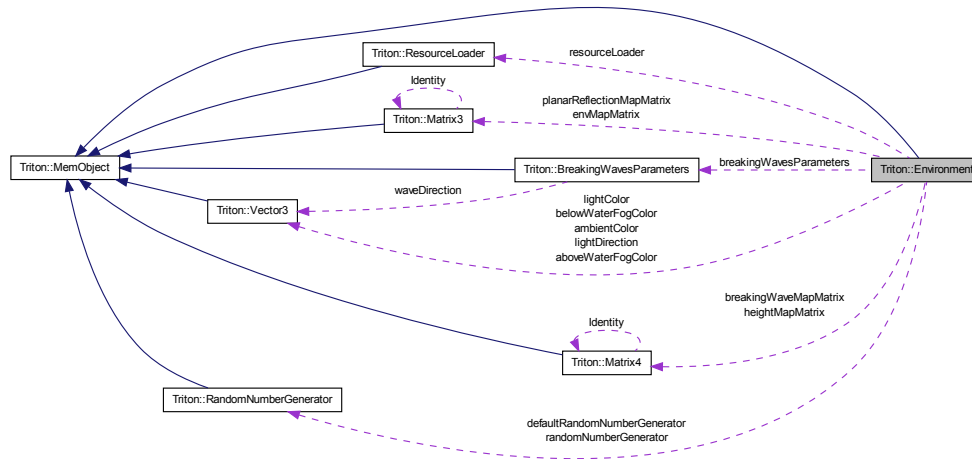
Triton's public interface for specifying the environmental conditions and camera properties.

```
#include <Environment.h>
```

Inheritance diagram for Triton::Environment:



Collaboration diagram for Triton::Environment:



Public Member Functions

- [Environment](#) ()
Constructor.
- `EnvironmentError` TRITONAPI [Initialize](#) (`CoordinateSystem` cs, `Renderer` ren, [ResourceLoader](#) *rl, `void` *device=NULL, `bool` hdr=false)
Initializes the environment prior to use.
- `virtual` [~Environment](#) ()
Virtual destructor.
- `void` TRITONAPI [SetLicenseCode](#) (`const char` *userName, `const char` *registrationCode)

Licensed users must call [SetLicenseCode](#) with your user name and registration code prior to using the [Environment](#) object.
- `void` TRITONAPI [SetRandomNumberGenerator](#) ([RandomNumberGenerator](#) *rng)

Set a custom [RandomNumberGenerator](#) - derived random number generator, to override Triton's default use of `stdlib`'s `rand()` function.
- [RandomNumberGenerator](#) *TRITONAPI [GetRandomNumberGenerator](#) () `const`

Returns either the default [RandomNumberGenerator](#) used for all random numbers in Triton, or a custom subclass of [RandomNumberGenerator](#) that was passed in via [Environment::SetRandomNumberGenerator\(\)](#).

- **ResourceLoader** *TRITONAPI **GetResourceLoader** () const
*Retrieves the **ResourceLoader** object passed in to the **Environment()** constructor.*
- void *TRITONAPI **GetDevice** () const
*Retrieves the DirectX device pointer passed in to the **Environment()** constructor, or NULL for OpenGL users.*
- void TRITONAPI **SetDirectionalLight** (const **Vector3** &direction, const **Vector3** &color)
Sets the color and direction of directional light used to light the water, as from the sun or moon.
- void TRITONAPI **SetAmbientLight** (const **Vector3** &color)
Sets the color of ambient light used to light the water, as from skylight.
- const **Vector3** &TRITONAPI **GetLightDirection** () const
*Retrieves the vector toward the infinitely distant directional light source passed in via **SetDirectionalLight()**.*
- const **Vector3** &TRITONAPI **GetDirectionalLightColor** () const
*Retrieves the RGB color of the directional light source passed in via **SetDirectionalLight()**.*
- const **Vector3** &TRITONAPI **GetAmbientLightColor** () const
*Retrieves the RGB color of the ambient light passed in via **SetAmbientLight()**.*
- void TRITONAPI **SetEnvironmentMap** (TextureHandle cubeMap, const **Matrix3** &textureMatrix=**Matrix3::Identity**)
Passes in an optional environment cube map used for rendering reflections in the water.
- TextureHandle TRITONAPI **GetEnvironmentMap** () const
*Retrieves the environment cube map passed in via **SetEnvironmentMap()**, which may be a GLuint, LPDIRECT3DCUBETEXTURE9, or ID3D11ShaderResourceView* depending on the renderer being used.*
- **Matrix3** TRITONAPI **GetEnvironmentMapMatrix** () const
*Retrieves the texture matrix used to transform the environment map lookups at runtime, which was optionally passed in via **SetEnvironmentMap()**.*
- void TRITONAPI **SetHeightMap** (TextureHandle pHeightMap, const **Matrix4** &worldToTextureCoords)
Optionally sets a height map used by Triton for improved water / shoreline interactions.
- TextureHandle TRITONAPI **GetHeightMap** () const

Retrieves the height map passed in via [SetHeightMap\(\)](#), which may be a *GLuint*, *LPDIRECT3DTEXTURE9*, or *ID3D11ShaderResourceView** depending on the renderer being used.

- **Matrix4** TRITONAPI [GetHeightMapMatrix](#) () const
Retrieves the texture matrix used to transform the height map lookups at runtime, which was passed in via [SetHeightMap\(\)](#).
- void TRITONAPI [SetBreakingWavesParameters](#) (const [BreakingWavesParameters](#) &params)
Configures the parameters used to simulate breaking waves at shorelines.
- const [BreakingWavesParameters](#) &TRITONAPI [GetBreakingWavesParameters](#) () const
Retrieves the current parameters for breaking waves.
- void TRITONAPI [SetPlanarReflectionMap](#) (TextureHandle textureMap, const [Matrix3](#) &textureMatrix, float normalDisplacementScale=0.125f)
Passes in an optional planar reflection map used for rendering local reflections in the water.
- TextureHandle TRITONAPI [GetPlanarReflectionMap](#) () const
Retrieves the environment cube map passed in via [SetPlanarReflectionMap\(\)](#), which may be a *GLuint*, *LPDIRECT3DTEXTURE9*, or *ID3D11ShaderResourceView** depending on the renderer being used.
- **Matrix3** TRITONAPI [GetPlanarReflectionMapMatrix](#) () const
Retrieves the texture matrix used to transform the planar reflection map lookups at runtime, which was passed in via [SetPlanarReflectionMap\(\)](#).
- float TRITONAPI [GetPlanarReflectionDisplacementScale](#) () const
Retrieves normal displacement scale set for planar reflections via [SetPlanarReflectionMap\(\)](#).
- void TRITONAPI [SimulateSeaState](#) (double beaufortScale, double windDirection, bool leftHanded=false)
Simulates a specific sea state on the Beaufort scale, by clearing out any existing wind fetches passed into the [Environment](#) and setting up a new one consistent with the state specified.
- void TRITONAPI [AddWindFetch](#) (const [WindFetch](#) &fetch, bool leftHanded=false)
Adds a [Triton::WindFetch](#) to the environment, which specifies an area of wind of a given speed and direction, which may or may not be localized.
- void TRITONAPI [ClearWindFetches](#) ()
Removes all wind fetches or sea state simulations from the simulated environment, resulting in a perfectly calm sea.

- void TRITONAPI [AddSwell](#) (float waveLength, float waveHeight, float direction, float phase=0, bool leftHanded=false)

Adds a swell wave to the ocean conditions in addition to the local wind waves.

- void TRITONAPI [ClearSwells](#) ()

Clears any swells previously added via [AddSwell\(\)](#).

- const [TRITON_VECTOR](#) ([SwellDescription](#))&TRITONAPI GetSwells() const

Retrieves the list of swells added via [AddSwell\(\)](#) following startup or the last call to [ClearSwells\(\)](#).

- void TRITONAPI [SetDouglasSeaScale](#) (int seaState, float windWaveDirection, int swellState, float swellDirection, bool leftHanded=false)

Simulate conditions as described by the Douglas sea scale (http://en.wikipedia.org/wiki/Douglas_Sea_Scale).

- void TRITONAPI [GetWind](#) (const [Vector3](#) &pos, double &windSpeed, double &windDirection, double &fetchLength) const

Computes the wind conditions at a given location, by evaluating all [WindFetch](#) objects that include the position.

- void TRITONAPI [SetSeaLevel](#) (double altitudeMSL)

If you want to change the mean sea level from a height of 0 in flat-earth coordinates, or from the WGS84 ellipsoid in geocentric coordinates, you may do so here.

- double TRITONAPI [GetSeaLevel](#) () const

Returns the offset for the mean sea level previously set by [SetSeaLevel\(\)](#).

- void TRITONAPI [SetAboveWaterVisibility](#) (double visibility, const [Vector3](#) &fog-Color)

Sets the simulated atmospheric visibility above the water, used to fog out the surface of the water when viewed from above.

- void TRITONAPI [GetAboveWaterVisibility](#) (double &visibility, [Vector3](#) &fog-Color) const

Retrieves the above-water visibility settings previously set with [SetAboveWaterVisibility\(\)](#).

- void TRITONAPI [SetBelowWaterVisibility](#) (double visibility, const [Vector3](#) &fog-Color)

Sets the simulated atmospheric visibility below the water, used to fog out the surface of the water when viewed from below.

- void TRITONAPI [GetBelowWaterVisibility](#) (double &visibility, [Vector3](#) &fog-Color) const

Retrieves the below-water visibility settings previously set with [SetBelowWaterVisibility\(\)](#).

- void TRITONAPI [SetSunIntensity](#) (float intensity)
Sets the intensity of the sunlight visible at the ocean surface; used to modulate the specular highlights of the sun on the water surface.
- float TRITONAPI [GetSunIntensity](#) () const
Retrieves the intensity of the transmitted direct sunlight on the water surface, as set with [SetSunIntensity\(\)](#).
- void TRITONAPI [SetWorldUnits](#) (double worldUnits)
Sets the size of one world unit in meters.
- double TRITONAPI [GetWorldUnits](#) () const
Retrieves the size of one world unit, in meters.
- CoordinateSystem TRITONAPI [GetCoordinateSystem](#) () const
Returns the CoordinateSystem passed into the [Environment\(\)](#) constructor, indicating the up vector and the presence of a geocentric or flat coordinate system.
- bool TRITONAPI [IsGeocentric](#) () const
Returns whether the CoordinateSystem passed into the [Environment\(\)](#) constructor is geocentric, indicating an elliptical or spherical coordinate system where all points are relative to the center of the Earth.
- Renderer TRITONAPI [GetRenderer](#) () const
Returns the Renderer specified in the [Environment\(\)](#) constructor, telling you what flavor of OpenGL or DirectX is being used to render Triton's water.
- bool TRITONAPI [IsOpenGL](#) () const
Returns whether the Renderer specified in the [Environment\(\)](#) constructor is an OpenGL renderer.
- bool TRITONAPI [IsDirectX](#) () const
Returns whether the Renderer specified in the [Environment\(\)](#) constructor is a DirectX renderer.
- void TRITONAPI [SetCameraMatrix](#) (const double *m)
Sets the modelview matrix used for rendering the ocean; this must be called every frame prior to calling [Ocean::Draw\(\)](#) if your camera orientation or position changes.
- void TRITONAPI [SetProjectionMatrix](#) (const double *p)
Sets the projection matrix used for rendering the ocean; this must be called every frame prior to calling [Ocean::Draw\(\)](#).
- void TRITONAPI [SetViewport](#) (int x, int y, int width, int height)
Informs Triton of the current viewport position and size.

- void TRITONAPI [SetZoomLevel](#) (float zoom)
If your camera is being zoomed from its typical field of view, use this method to let Triton know about the zoom factor.
- float TRITONAPI [GetZoomLevel](#) () const
Retrieves any zoom level previously set with [SetZoomLevel\(\)](#), or 1.0 if default.
- void TRITONAPI [SetUserDefinedVertString](#) (const char *userDefinedString)
Sets a user defined string to be prepended to all vertex shaders.
- void TRITONAPI [SetUserDefinedFragString](#) (const char *userDefinedString)
Sets a user defined string to be prepended to all fragment shaders.
- const char *TRITONAPI [GetUserDefinedVertString](#) () const
Retrieves the user defined vertex string previously set with [SetUserDefinedVertString](#).
- const char *TRITONAPI [GetUserDefinedFragString](#) () const
Retrieves the user defined fragment string previously set with [SetUserDefinedFragString](#).
- bool TRITONAPI [GetViewport](#) (int &x, int &y, int &width, int &height) const
Retrieves any viewport information previously set via [SetViewport\(\)](#).
- const double *TRITONAPI [GetCameraMatrix](#) () const
Retrieves an array of 16 doubles representing the modelview matrix passed in via [SetCameraMatrix\(\)](#).
- const double *TRITONAPI [GetProjectionMatrix](#) () const
Retrieves an array of 16 doubles representing the projection matrix passed in via [SetProjectionMatrix\(\)](#).
- const double *TRITONAPI [GetCameraPosition](#) () const
Retrieves an array of 3 doubles representing the X, Y, and Z position of the camera, extracted from the modelview matrix passed in via [SetCameraMatrix\(\)](#).
- [Vector3](#) TRITONAPI [GetUpVector](#) () const
Retrieves a normalized vector pointing "up", based on the coordinate system specified in [Environment::Initialize\(\)](#) and the current position from the modelview matrix passed in through [Environment::SetCameraMatrix\(\)](#).
- [Vector3](#) TRITONAPI [GetRightVector](#) () const
Retrieves a normalized vector pointing "right", based on the coordinate system specified in [Environment::Initialize\(\)](#) and the current position from the modelview matrix passed in through [Environment::SetCameraMatrix\(\)](#).
- void TRITONAPI [SetConfigOption](#) (const char *key, const char *value)

Sets a configuration setting (defaults in resources/triton.config.) Many settings are read at initialization, so call this as early as possible after initializing the [Environment](#).

- `const char *TRITONAPI GetConfigOption (const char *key)`
Retrieves the configuration setting for the given configuration key, as set in resources/triton.config or via [SetConfigOption\(\)](#).
- `bool TRITONAPI CullSphere (const Vector3 &position, double radius) const`
Returns true if the given sphere lies within the view frustum, as defined by the modelview - projection matrix passed in via [SetCameraMatrix\(\)](#) and [SetProjectionMatrix\(\)](#).
- `const bool TRITONAPI GetHDREnabled () const`
Retrieves whether HDR mode is enabled, indicating whether color values are clamped to [0,1.0], as set in [Environment::Initialize\(\)](#)
- `void TRITONAPI EnableOpenMP (bool enabled)`
Sets Triton's usage of OpenMP to enable parallel processing of CPU-intensive tasks across multiple CPU cores.
- `bool TRITONAPI GetOpenMPEnabled () const`
Retrieves whether OpenMP has been enabled to take advantage of multi-core CPU's.
- `float TRITONAPI GetMaximumWaveHeight () const`
Gets the estimated maximum wave height in meters at the camera position, given the current wind and swell conditions.

22.3.1 Detailed Description

Triton's public interface for specifying the environmental conditions and camera properties. The [Ocean](#) constructor requires an [Environment](#) object, so you'll need to create this first.

22.3.2 Constructor & Destructor Documentation

22.3.2.1 `Triton::Environment::Environment ( )`

Constructor.

22.3.2.2 `virtual Triton::Environment::~~Environment ( ) [virtual]`

Virtual destructor.

22.3.3 Member Function Documentation

22.3.3.1 `void TRITONAPI Triton::Environment::AddSwell ( float waveLength, float waveHeight, float direction, float phase = 0, bool leftHanded = false )`

Adds a swell wave to the ocean conditions in addition to the local wind waves.

Swells may sometimes originate from distant storms and not have anything to do with the local conditions. You may add as many swells as you wish; they incur no run-time performance impact. Swells are not supported in the GERSTNER water model.

See also

[ClearSwells\(\)](#)

Parameters

<i>waveLength</i>	The wavelength of the swell wave, from peak to peak, in world units. Swells are generally around 100-200 m.
<i>waveHeight</i>	The swell wave height, from peak to trough, in world units.
<i>direction</i>	The direction of the swell wave, in radians. This could be different from the local wind direction.
<i>phase</i>	The phase offset of the swell wave, in radians.
<i>leftHanded</i>	If you are using a left-handed coordinate system (for example, Y is "up" but positive Z is "North"), pass true in order to ensure your wave direction is represented correctly.

22.3.3.2 `void TRITONAPI Triton::Environment::AddWindFetch ( const WindFetch & fetch, bool leftHanded = false )`

Adds a [Triton::WindFetch](#) to the environment, which specifies an area of wind of a given speed and direction, which may or may not be localized.

This [WindFetch](#) will be added to any other wind passed in previously, unless [ClearWindFetches\(\)](#) is called first.

All waves in Triton are a result of simulated wind conditions. Without wind, there will be no waves. Stronger winds traveling across longer distances will result in higher waves.

Parameters

<i>fetch</i>	The WindFetch to add to the simulated environment.
<i>leftHanded</i>	If you are using a left-handed coordinate system (for example, Y is "up" but positive Z is "North"), pass true in order to ensure your wave direction is represented correctly.

22.3.3.3 `void TRITONAPI Triton::Environment::ClearSwells ( )`

Clears any swells previously added via [AddSwell\(\)](#).

See also

[AddSwell\(\)](#)

22.3.3.4 void TRITONAPI Triton::Environment::ClearWindFetches ()

Removes all wind fetches or sea state simulations from the simulated environment, resulting in a perfectly calm sea.

Call [AddWindFetch\(\)](#) or [SimulateSeaState\(\)](#) to add wind back in and generate waves as a result.

22.3.3.5 bool TRITONAPI Triton::Environment::CullSphere (const Vector3 & *position*, double *radius*) const

Returns true if the given sphere lies within the view frustum, as defined by the modelview - projection matrix passed in via [SetCameraMatrix\(\)](#) and [SetProjectionMatrix\(\)](#).

Parameters

<i>position</i>	The center of the sphere in world coordinates.
<i>radius</i>	The radius of the sphere in world coordinates.

Returns

True if the sphere is not visible and should be culled.

22.3.3.6 void TRITONAPI Triton::Environment::EnableOpenMP (bool *enabled*) [inline]

Sets Triton's usage of OpenMP to enable parallel processing of CPU-intensive tasks across multiple CPU cores.

You might want to disable this if you are more concerned about limiting CPU usage than maintaining the fastest possible ocean rendering. OpenMP is enabled by default. Note, if the IPP FFT implementation is being used instead of CUDA or OpenCL, multi-core usage will happen regardless of this setting.

22.3.3.7 void TRITONAPI Triton::Environment::GetAboveWaterVisibility (double & *visibility*, Vector3 & *fogColor*) const [inline]

Retrieves the above-water visibility settings previously set with [SetAboveWaterVisibility\(\)](#).

Parameters

<i>visibility</i>	Receives the visibility, in world units, above the water.
<i>fogColor</i>	Receives the fog color, in normalized RGB units.

22.3.3.8 `const Vector3& TRITONAPI Triton::Environment::GetAmbientLightColor ( ) const`
`[inline]`

Retrieves the RGB color of the ambient light passed in via [SetAmbientLight\(\)](#).

22.3.3.9 `void TRITONAPI Triton::Environment::GetBelowWaterVisibility ( double & visibility,`
`Vector3 & fogColor ) const [inline]`

Retrieves the below-water visibility settings previously set with [SetBelowWaterVisibility\(\)](#).

Parameters

<i>visibility</i>	Receives the visibility, in world units, above the water.
<i>fogColor</i>	Receives the fog color, in normalized RGB units.

22.3.3.10 `const BreakingWavesParameters& TRITONAPI`
`Triton::Environment::GetBreakingWavesParameters ( ) const [inline]`

Retrieves the current parameters for breaking waves.

See also

[Environment::SetBreakingWavesParameters\(\)](#)

22.3.3.11 `const double* TRITONAPI Triton::Environment::GetCameraMatrix ( ) const`
`[inline]`

Retrieves an array of 16 doubles representing the modelview matrix passed in via [SetCameraMatrix\(\)](#).

22.3.3.12 `const double* TRITONAPI Triton::Environment::GetCameraPosition ( ) const`
`[inline]`

Retrieves an array of 3 doubles representing the X, Y, and Z position of the camera, extracted from the modelview matrix passed in via [SetCameraMatrix\(\)](#).

22.3.3.13 `const char* TRITONAPI Triton::Environment::GetConfigOption ( const char * key )`

Retrieves the configuration setting for the given configuration key, as set in resources/triton.config or via [SetConfigOption\(\)](#).

Parameters

<i>key</i>	The configuration key to retrieve
------------	-----------------------------------

Returns

The value of the key specified, or NULL if not found.

22.3.3.14 `CoordinateSystem TRITONAPI Triton::Environment::GetCoordinateSystem ( ) const`
[inline]

Returns the `CoordinateSystem` passed into the [Environment\(\)](#) constructor, indicating the up vector and the presence of a geocentric or flat coordinate system.

22.3.3.15 `void* TRITONAPI Triton::Environment::GetDevice ( ) const` [inline]

Retrieves the DirectX device pointer passed in to the [Environment\(\)](#) constructor, or NULL for OpenGL users.

22.3.3.16 `const Vector3& TRITONAPI Triton::Environment::GetDirectionalLightColor ( ) const`
[inline]

Retrieves the RGB color of the directional light source passed in via [SetDirectionalLight\(\)](#).

22.3.3.17 `TextureHandle TRITONAPI Triton::Environment::GetEnvironmentMap ( ) const`
[inline]

Retrieves the environment cube map passed in via [SetEnvironmentMap\(\)](#), which may be a `GLuint`, `LPDIRECT3DCUBETEXTURE9`, or `ID3D11ShaderResourceView*` depending on the renderer being used.

22.3.3.18 `Matrix3 TRITONAPI Triton::Environment::GetEnvironmentMapMatrix ( ) const`
[inline]

Retrieves the texture matrix used to transform the environment map lookups at runtime, which was optionally passed in via [SetEnvironmentMap\(\)](#).

22.3.3.19 `TextureHandle TRITONAPI Triton::Environment::GetHeightMap ( ) const`
[inline]

Retrieves the height map passed in via [SetHeightMap\(\)](#), which may be a `GLuint`, `LPDIRECT3DTEXTURE9`, or `ID3D11ShaderResourceView*` depending on the renderer being used.

22.3.3.20 **Matrix4** TRITONAPI Triton::Environment::GetHeightMapMatrix () const
[inline]

Retrieves the texture matrix used to transform the height map lookups at runtime, which was passed in via [SetHeightMap\(\)](#).

This matrix transforms world coordinates into height map texture coordinates.

22.3.3.21 **const Vector3&** TRITONAPI Triton::Environment::GetLightDirection () const
[inline]

Retrieves the vector toward the infinitely distant directional light source passed in via [SetDirectionalLight\(\)](#).

22.3.3.22 **float** TRITONAPI Triton::Environment::GetMaximumWaveHeight () const

Gets the estimated maximum wave height in meters at the camera position, given the current wind and swell conditions.

22.3.3.23 **bool** TRITONAPI Triton::Environment::GetOpenMPEnabled () const [inline]

Retrieves whether OpenMP has been enabled to take advantage of multi-core CPU's.

See also

[EnableOpenMP\(\)](#)

22.3.3.24 **float** TRITONAPI Triton::Environment::GetPlanarReflectionDisplacementScale ()
const [inline]

Retrieves normal displacement scale set for planar reflections via [SetPlanarReflectionMap\(\)](#).

22.3.3.25 **TextureHandle** TRITONAPI Triton::Environment::GetPlanarReflectionMap () const
[inline]

Retrieves the environment cube map passed in via [SetPlanarReflectionMap\(\)](#), which may be a GLuint, LPDIRECT3DTEXTURE9, or ID3D11ShaderResourceView* depending on the renderer being used.

22.3.3.26 **Matrix3** TRITONAPI Triton::Environment::GetPlanarReflectionMapMatrix () const
[inline]

Retrieves the texture matrix used to transform the planar reflection map lookups at runtime, which was passed in via [SetPlanarReflectionMap\(\)](#).

22.3.3.27 `const double* TRITONAPI Triton::Environment::GetProjectionMatrix ( ) const`
`[inline]`

Retrieves an array of 16 doubles representing the projection matrix passed in via [SetProjectionMatrix\(\)](#).

22.3.3.28 `RandomNumberGenerator* TRITONAPI Triton::Environment::GetRandomNumberGenerator ( ) const` `[inline]`

Returns either the default [RandomNumberGenerator](#) used for all random numbers in Triton, or a custom subclass of [RandomNumberGenerator](#) that was passed in via [Environment::SetRandomNumberGenerator\(\)](#).

Returns

The [RandomNumberGenerator](#) instance in use by Triton.

22.3.3.29 `Renderer TRITONAPI Triton::Environment::GetRenderer ( ) const` `[inline]`

Returns the Renderer specified in the [Environment\(\)](#) constructor, telling you what flavor of OpenGL or DirectX is being used to render Triton's water.

22.3.3.30 `ResourceLoader* TRITONAPI Triton::Environment::GetResourceLoader ( ) const`
`[inline]`

Retrieves the [ResourceLoader](#) object passed in to the [Environment\(\)](#) constructor.

22.3.3.31 `Vector3 TRITONAPI Triton::Environment::GetRightVector ( ) const`

Retrieves a normalized vector pointing "right", based on the coordinate system specified in [Environment::Initialize\(\)](#) and the current position from the modelview matrix passed in through [Environment::SetCameraMatrix\(\)](#).

22.3.3.32 `double TRITONAPI Triton::Environment::GetSeaLevel ( ) const` `[inline]`

Returns the offset for the mean sea level previously set by [SetSeaLevel\(\)](#).

Returns

The offset in world units that the mean sea level is displaced by.

22.3.3.33 `float TRITONAPI Triton::Environment::GetSunIntensity ( ) const` `[inline]`

Retrieves the intensity of the transmitted direct sunlight on the water surface, as set with [SetSunIntensity\(\)](#).

22.3.3.34 Vector3 TRITONAPI Triton::Environment::GetUpVector () const

Retrieves a normalized vector pointing "up", based on the coordinate system specified in [Environment::Initialize\(\)](#) and the current position from the modelview matrix passed in through [Environment::SetCameraMatrix\(\)](#).

22.3.3.35 bool TRITONAPI Triton::Environment::GetViewport (int & x, int & y, int & width, int & height) const

Retrieves any viewport information previously set via [SetViewport\(\)](#).

If [SetViewport\(\)](#) has not been called, this method will return false and return zeros for all parameters.

Parameters

<i>x</i>	The x position of the current viewport origin.
<i>y</i>	The y position of the current viewport origin.
<i>width</i>	The width of the current viewport.
<i>height</i>	The height of the current viewport.

Returns

true if SetViewport was previously called, and valid information is returned.

22.3.3.36 void TRITONAPI Triton::Environment::GetWind (const Vector3 & pos, double & windSpeed, double & windDirection, double & fetchLength) const

Computes the wind conditions at a given location, by evaluating all [WindFetch](#) objects that include the position.

See also

[AddWindFetch\(\)](#)
[SimulateSeaState\(\)](#)

Parameters

<i>pos</i>	A const reference to the position at which wind conditions should be computed.
<i>windSpeed</i>	A reference to a double that will receive the overall wind speed at this location, in world units per second.
<i>windDirection</i>	A reference to a double that will receive the overall wind direction at this location, in radians.
<i>fetchLength</i>	A reference to a double that will receive the fetch length of the farthest WindFetch affecting this location. May return 0 if the local WindFetches have no fetch length specified via WindFetch::SetLocalization() or WindFetch::SetFetchLength() .

22.3.3.37 `double TRITONAPI Triton::Environment::GetWorldUnits ( ) const [inline]`

Retrieves the size of one world unit, in meters.

See also

[SetWorldUnits\(\)](#)

22.3.3.38 `float TRITONAPI Triton::Environment::GetZoomLevel ( ) const [inline]`

Retrieves any zoom level previously set with [SetZoomLevel\(\)](#), or 1.0 if default.

Returns

The zoom level specified by [Environment::SetZoomLevel\(\)](#) if any, 1.0 otherwise.

22.3.3.39 `EnvironmentError TRITONAPI Triton::Environment::Initialize ( CoordinateSystem cs, Renderer ren, ResourceLoader * rl, void * device = NULL, bool hdr = false )`

Initializes the environment prior to use.

Parameters

<i>cs</i>	The CoordinateSystem your application is using, which may be a geocentric system based on an elliptical WGS84 or spherical earth model, or a flat cartesian model, with up pointing along either the Y or Z axes.
<i>ren</i>	Specifies the version of OpenGL or DirectX your application is using. Triton will render the ocean using the same graphics subsystem.
<i>rl</i>	A ResourceLoader object that is used by Triton for loading its graphics, shader, and configuration resources. This may be an instance of Triton's default ResourceLoader class that loads loose files in Triton's resources directory directly from disk, or your own derived class that handles resource management in some other way.
<i>device</i>	Unused for OpenGL contexts. For DirectX users, this must be a pointer to your valid and initialized DIRECT3DDEVICE9 or ID3D11Device.
<i>hdr</i>	Whether High Dynamic Range rendering is desired, meaning colors will not be clamped to [0.0,1.0]. If true, HDR lighting values passed in via Environment::SetDirectionalLight() , Environment::SetAmbientLight() , and/or via floating point reflection or environment maps will be preserved.

Returns

An error code if the environment failed to initialize, in which case you can't use this environment. To receive more details on why it failed, enable the setting `enable-debug-messages` in `resources/triton.config` which will send more info to your debugger's output. If initialization succeeded, you'll get back `SUCCEEDED`.

22.3.3.40 `bool TRITONAPI Triton::Environment::IsDirectX ( ) const` `[inline]`

Returns whether the Renderer specified in the [Environment\(\)](#) constructor is a DirectX renderer.

22.3.3.41 `bool TRITONAPI Triton::Environment::IsGeocentric ( ) const` `[inline]`

Returns whether the CoordinateSystem passed into the [Environment\(\)](#) constructor is geocentric, indicating an elliptical or spherical coordinate system where all points are relative to the center of the Earth.

22.3.3.42 `bool TRITONAPI Triton::Environment::IsOpenGL ( ) const` `[inline]`

Returns whether the Renderer specified in the [Environment\(\)](#) constructor is an OpenGL renderer.

22.3.3.43 `void TRITONAPI Triton::Environment::SetAboveWaterVisibility ( double visibility, const Vector3 & fogColor )` `[inline]`

Sets the simulated atmospheric visibility above the water, used to fog out the surface of the water when viewed from above.

You may use this method to fog the ocean consistently with other objects in your scene.

The visibility specified will be transformed into an exponential fog extinction value using the Koschmieder equation: $visibility = 3.912 / extinction$

Parameters

<i>visibility</i>	The visibility, in world units, above the water.
<i>fogColor</i>	The fog color, in normalized RGB units.

22.3.3.44 `void TRITONAPI Triton::Environment::SetAmbientLight ( const Vector3 & color )` `[inline]`

Sets the color of ambient light used to light the water, as from skylight.

Parameters

<i>color</i>	the RGB color of the ambient light.
--------------	-------------------------------------

22.3.3.45 `void TRITONAPI Triton::Environment::SetBelowWaterVisibility ( double visibility, const Vector3 & fogColor )` `[inline]`

Sets the simulated atmospheric visibility below the water, used to fog out the surface of the water when viewed from below.

You may use this method to fog the ocean consistently with other objects in your scene. While underwater, the application is responsible for clearing the back buffer to the fog color to match the color passed in here.

The visibility specified will be transformed into an exponential fog extinction value using the Koschmieder equation: $\text{visibility} = 3.912 / \text{extinction}$

Parameters

<i>visibility</i>	The visibility, in world units, below the water.
<i>fogColor</i>	The fog color, in normalized RGB units.

22.3.3.46 void TRITONAPI Triton::Environment::SetBreakingWavesParameters (const BreakingWavesParameters & params) [inline]

Configures the parameters used to simulate breaking waves at shorelines.

This will only have an effect if you created your [Ocean](#) object with the enableBreakingWaves parameter of `Ocean::Ocean()` set to true, and if you have also passed in a height map containing bathymetry information via [Environment::SetHeightMap\(\)](#). Please see the documentation for [Triton::BreakingWavesParameters](#) for a description of the various settings. Generally, you'll be OK just setting the direction of the waves to point toward the nearest shoreline, and leave the rest of the settings at their defaults.

See also

`GetBreakingWavesParameters()`

22.3.3.47 void TRITONAPI Triton::Environment::SetCameraMatrix (const double * m)

Sets the modelview matrix used for rendering the ocean; this must be called every frame prior to calling [Ocean::Draw\(\)](#) if your camera orientation or position changes.

Parameters

<i>m</i>	A pointer to 16 doubles representing a 4x4 modelview matrix.
----------	--

22.3.3.48 void TRITONAPI Triton::Environment::SetConfigOption (const char * key, const char * value)

Sets a configuration setting (defaults in resources/triton.config.) Many settings are read at initialization, so call this as early as possible after initializing the [Environment](#).

Parameters

<i>key</i>	The configuration entry name to modify
<i>value</i>	The value to set this entry to.

22.3.3.49 void TRITONAPI Triton::Environment::SetDirectionalLight (const Vector3 & direction, const Vector3 & color)

Sets the color and direction of directional light used to light the water, as from the sun or moon.

Take care that the direction points toward the light source, and not from it - invalid coloration of the water will result if this direction is negated.

Parameters

<i>direction</i>	A normalized vector pointing toward the infinitely distant light source.
<i>color</i>	The RGB color of the light.

22.3.3.50 void TRITONAPI Triton::Environment::SetDouglasSeaScale (int seaState, float windWaveDirection, int swellState, float swellDirection, bool leftHanded = false)

Simulate conditions as described by the Douglas sea scale (http://en.wikipedia.org/wiki/Douglas_Sea_Scale).

This will clear out any previously set WindFetches, Beaufort scale setting from [SimulateSeaState\(\)](#), and swells created with [AddSwell\(\)](#). Swell simulation is not supported with the GERSTNER water model.

Parameters

<i>seaState</i>	A value from 0 to 9, 0 describing "calm and glassy" and 9 describing "phenomenal" conditions.
<i>wind-WaveDirection</i>	The direction of wind waves in radians.
<i>swellState</i>	A value from 0 to 9 describing high-wavelength swell waves, 0 describing no swell and 9 describing "confused" seas.
<i>swellDirection</i>	The direction of the swell waves in radians.
<i>leftHanded</i>	If you are using a left-handed coordinate system (for example, Y is "up" but positive Z is "North"), pass true in order to ensure your wave direction is represented correctly.

22.3.3.51 void TRITONAPI Triton::Environment::SetEnvironmentMap (TextureHandle cubeMap, const Matrix3 & textureMatrix = Matrix3::Identity) [inline]

Passes in an optional environment cube map used for rendering reflections in the water.

If unused, Triton will instead reflect a constant color based on the ambient light passed in via [SetAmbientLight\(\)](#). The caller is responsible for releasing or deleting this resource at shutdown.

See also

[SetEnvironmentMap\(\)](#)

Parameters

<i>cubeMap</i>	A cube map texture resource, which should be cast to a TextureHandle. Under OpenGL, this must be a GLuint indicating the ID of the GL_TEXTURE_CUBE_MAP returned from glGenTextures. Under DirectX9, this must be a LPDIRECT3DCUBETEXTURE9. Under DirectX11, this must be a ID3D11ShaderResourceView pointer with an underlying ViewDimension of D3D11_SRV_DIMENSION_TEXTURECUBE.
<i>textureMatrix</i>	An optional texture matrix used to transform the 3D coordinates used to access the cube map. If your cube map isn't oriented with the same cartesian axes used by your simulation, you can use this parameter to account for any differences in your cube map's coordinate system and your simulation's. When using DirectX, it's likely that you will need to use this matrix to flip the cube map upside-down due to DirectX's left-handed convention. For example, if Y is "up", pass a scaling matrix to scale Y by -1 if reflections seem to be coming from the bottom of the environment map instead of from the top.

22.3.3.52 void TRITONAPI Triton::Environment::SetHeightMap (TextureHandle pHeightMap, const Matrix4 & worldToTextureCoords)

Optionally sets a height map used by Triton for improved water / shoreline interactions.

If a height map is provided that includes bathymetry data - that is, it extends below sea level to include the surface of the sea floor - Triton can use this to obtain the depth of the water at each point. This is used for transparency effects and breaking waves at shorelines, and to prevent Triton from drawing water over the terrain.

Take care to only call this method when your height map's content changes. Triton must make a system memory copy of it for height queries, which impacts performance.

For accurate water surface height queries near the shore, DirectX9 users must provide a lockable texture (meaning it is not created in the default pool, or it is created with render target usage,) in D3DFMT_R32F format. DirectX11 users must use format DXGI_FORMAT_R32_FLOAT. OpenGL users may use any floating-point format, with the height in the red or luminance channel, such as GL_LUMINANCE32F_ARB.

If no height map is provided, you can instead use [Triton::Ocean::SetDepth\(\)](#) to specify a uniformly sloping sea floor from the current camera location, and the depth buffer will be used to properly sort terrain against the water.

See also

[Environment::SetBreakingWavesParameters\(\)](#)

Parameters

<i>pHeightMap</i>	Under OpenGL, this must be a GLuint indicating the ID of the GL_TEXTURE_2D returned from glGenTextures. Under DirectX9, this must be a LPDIRECT3DTEXTURE9. Under DirectX11, this must be a ID3D11ShaderResourceView pointer with an underlying ViewDimension of D3D11_SRV_DIMENSION_TEXTURE2D. This texture is expected to contain a single 16 or 32-bit-per-component floating-point channel representing the height at each point, in world units. Pass NULL to disable any previously set height map.
<i>worldToTextureCoords</i>	A matrix to transform world coordinates to texture coordinates in the height map. Generally this is an orthographic matrix looking down at the height field, scaled and translated into texture coordinate space.

22.3.3.53 `void TRITONAPI Triton::Environment::SetLicenseCode (const char * userName,
const char * registrationCode)`

Licensed users must call SetLicenseCode with your user name and registration code prior to using the [Environment](#) object.

Visit <http://www.sundog-soft.com/> to purchase a license. If you don't call SetLicenseCode or pass invalid parameters to it, Triton will run in evaluation mode, which will terminate your application after five minutes of runtime.

Parameters

<i>userName</i>	The user name given to you with your license purchase.
<i>registrationCode</i>	The registration code given to you with your license purchase.

22.3.3.54 `void TRITONAPI Triton::Environment::SetPlanarReflectionMap (TextureHandle
textureMap, const Matrix3 & textureMatrix, float normalDisplacementScale =
0.125f) [inline]`

Passes in an optional planar reflection map used for rendering local reflections in the water.

Triton can use planar reflection map & environment map together. Alpha channel in planar reflection map is used to blend between planar reflection & environment map. If planar reflection is not used Triton falls back to environment map (it is the same result as if planar reflection had 0 on alpha in every texel).

See also

[SetEnvironmentMap\(\)](#)
[Triton::Ocean::SetPlanarReflectionBlend\(\)](#)
[Triton::Ocean::ComputeReflectionMatrices\(\)](#)

Parameters

<i>textureMap</i>	A 2D map texture resource, which should be cast to a TextureHandle. Under OpenGL, this must be a GLuint indicating the ID of the GL_TEXTURE_2D returned from glGenTextures. Under DirectX9, this must be a LPDIRECT3DTEXTURE9. Under DirectX11, this must be a ID3D11ShaderResourceView pointer with an underlying ViewDimension of D3D11_SRV_DIMENSION_TEXTURE2D.
<i>textureMatrix</i>	A required texture matrix used to project the vector computed by triton to reflection map texture coordinates. Generally, this will be the main scene's view rotation matrix * projection matrix, multiplied by a translation of (1, 1, 1) and then by a scale of (0.5, 0.5, 0.5) to transform normalized device coordinates to texture coordinates. Triton's "Input" is a view vector perturbed by normal.xy components. Such a vector approximates wave reflection wiggle and can be used to directly access planar reflection map. See description of parameter normalDisplacementScale to learn why reflection vector cannot be used directly to access the planar reflection map. The input Vector passed to textureMatrix will be defined in world space coordinates translated to the view point. In other words this coordinate space has the same orientation as world space but its origin (point 0,0,0) is moved to the camera location. This is the same coordinate space that env map projection uses. Advanced users may see how reflection (P) variable is handled in Triton pixel (fragment) shaders.
<i>normalDisplacementScale</i>	A scale factor used to perturb vertex by normal.xy to get more realistic reflection from rough waves. Range of reasonable values is 0..4. Default is 0.125 Realtime bumpy surface reflection approaches are usually based on approximation of reflection from flat surface. However, method of reflection based on planar projection has serious limitation. It assumes that view vector (and reflection vector) angle strictly corresponds to the incident point on the surface where vector was reflected. If surface is not perfectly flat and water waves are of course an example of such surface, above assumption fails and many points at the ocean surface can reflect vectors at the same direction. If such reflection was used to address a planar map texel we could see reflections of the objects at random points on the surface. To avoid such effect, usually some limits are imposed on reflected vector or reflected texture coords. Triton adopts classic approach to the above problem which works by actually using a view vector perturbed by an offset computed from normal.xy scaled by normalDisplacementScale. Since normal.xy components are never larger than unit value we can be sure that reflection vector will fit in finite margin defined by normalDisplacementScale.

22.3.3.55 void TRITONAPI Triton::Environment::SetProjectionMatrix (const double * p)

Sets the projection matrix used for rendering the ocean; this must be called every frame prior to calling [Ocean::Draw\(\)](#).

Parameters

<i>p</i>	A pointer to 16 doubles representing a 4x4 projection matrix.
----------	---

22.3.3.56 void TRITONAPI Triton::Environment::SetRandomNumberGenerator (RandomNumberGenerator * *rng*)

Set a custom [RandomNumberGenerator](#) - derived random number generator, to override Triton's default use of `stdlib's rand()` function.

This may be useful for ensuring deterministic behavior across channels (although a simpler approach may be calling `srand()` with a consistent seed from your application.) If this method is not called, a default random number generator will be used automatically.

Parameters

<i>rng</i>	An instance of a class derived from RandomNumberGenerator that will handle all random number generation within Triton.
------------	--

22.3.3.57 void TRITONAPI Triton::Environment::SetSeaLevel (double *altitudeMSL*) [inline]

If you want to change the mean sea level from a height of 0 in flat-earth coordinates, or from the WGS84 ellipsoid in geocentric coordinates, you may do so here.

See also

[GetSeaLevel\(\)](#)

Parameters

<i>altitudeMSL</i>	The offset in world units to displace mean sea level by.
--------------------	--

22.3.3.58 void TRITONAPI Triton::Environment::SetSunIntensity (float *intensity*) [inline]

Sets the intensity of the sunlight visible at the ocean surface; used to modulate the specular highlights of the sun on the water surface.

Normally this is 1.0, but you might want to decrease it for example if the sun is obscured by clouds.

22.3.3.59 void TRITONAPI Triton::Environment::SetViewport (int *x*, int *y*, int *width*, int *height*)

Informs Triton of the current viewport position and size.

Calling this is optional, but allows Triton to avoid querying OpenGL or DirectX for the current viewport parameters, which can cause a pipeline stall. If you call this method, you are responsible for calling it whenever the viewport changes.

Parameters

<i>x</i>	The x position of the current viewport origin.
<i>y</i>	The y position of the current viewport origin.
<i>width</i>	The width of the current viewport.
<i>height</i>	The height of the current viewport.

22.3.3.60 `void TRITONAPI Triton::Environment::SetWorldUnits ( double worldUnits )`
`[inline]`

Sets the size of one world unit in meters.

By default, one world unit is assumed to mean one meter. If this is not the case for your coordinate system, be sure to call [SetWorldUnits\(\)](#) immediately after instantiating your [Environment](#) class.

22.3.3.61 `void TRITONAPI Triton::Environment::SetZoomLevel ( float zoom )` `[inline]`

If your camera is being zoomed from its typical field of view, use this method to let Triton know about the zoom factor.

This will automatically adjust things like LOD switch distances, noise blending distances, etc. to ensure the ocean looks as you would expect when zoomed in. This method does NOT modify the projection matrix you passed in with [SetProjectionMatrix\(\)](#), so it will NOT actually change the field of view of the ocean rendering - that's up to you when constructing the projection matrix.

Parameters

<i>zoom</i>	The zoom level of the current camera, if any. 1.0 represents no zoom, 10.0 would represent 10X.
-------------	---

22.3.3.62 `void TRITONAPI Triton::Environment::SimulateSeaState ( double beaufortScale, double windDirection, bool leftHanded = false )`

Simulates a specific sea state on the Beaufort scale, by clearing out any existing wind fetches passed into the [Environment](#) and setting up a new one consistent with the state specified.

Any subsequent calls to [AddWindFetch\(\)](#) will create wind additive to that created for the given sea state, so be sure to call [ClearWindFetches\(\)](#) if you intend to mix and match calls to [SimulateSeaState\(\)](#) and [AddWindFetch\(\)](#).

See http://en.wikipedia.org/wiki/Beaufort_scale for detailed descriptions of Beaufort numbers and the wave conditions they specify. At a high level,

0: Calm 1: Light air 2: Light breeze 3: Gentle breeze 4: Moderate breeze 5: Fresh breeze 6: Strong breeze 7: High wind 8: Gale 9: Storm 10: Strong Storm 11: Violent Storm 12: Hurricane

Parameters

<i>beaufortScale</i>	The Beaufort scale number specifying the desired wind and sea conditions. This may be a floating point value.
<i>windDirection</i>	The direction of the wind, specified in radians.
<i>leftHanded</i>	If you are using a left-handed coordinate system (for example, Y is "up" but positive Z is "North"), pass true in order to ensure your wave direction is represented correctly.

22.3.3.63 const Triton::Environment::TRITON_VECTOR (SwellDescription) const

Retrieves the list of swells added via [AddSwell\(\)](#) following startup or the last call to [ClearSwells\(\)](#).

See also

[AddSwell\(\)](#).

The documentation for this class was generated from the following file:

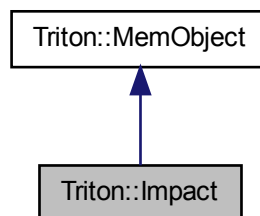
- C:/triton/trunk/Public Headers/[Environment.h](#)

22.4 Triton::Impact Class Reference

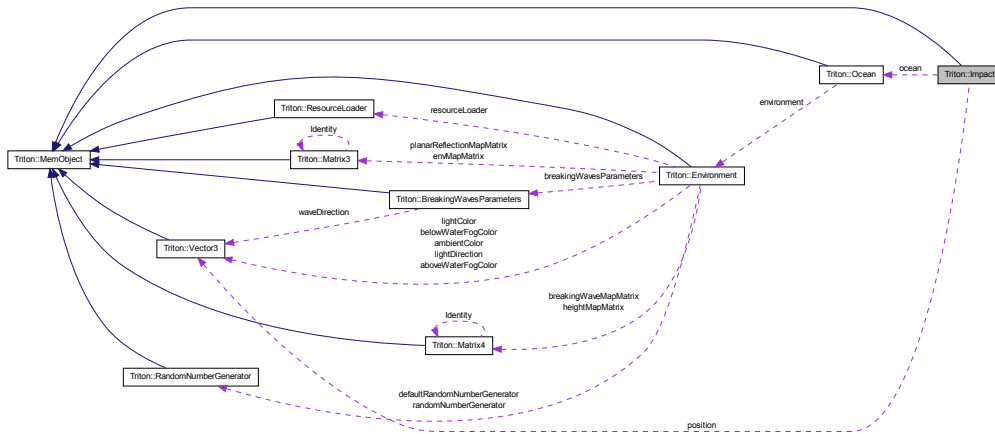
A [RotorWash](#) object will generate spray and circular waves on the ocean surface in the direction it is pointing.

```
#include <Impact.h>
```

Inheritance diagram for Triton::Impact:



Collaboration diagram for Triton::Impact:



Public Member Functions

- **Impact** (**Ocean** *pOcean, double pImpactorDiameter, double pMass, bool pSprayEffects=false, double pSprayScale=2.0)

Construct a **Impact** with the **Triton::Ocean** it will be associated with.

- void TRITONAPI **Trigger** (const **Vector3** &pPosition, const **Vector3** &pDirection, double pVelocity, double pTime)

Starts the effect of an impact at a given position, direction, and velocity.

- **Vector3** TRITONAPI **GetPosition** () const

Retrieves the world position of the **RotorWash**.

- double TRITONAPI **GetVelocity** () const

Retrieves the velocity of the **RotorWash**.

22.4.1 Detailed Description

A **RotorWash** object will generate spray and circular waves on the ocean surface in the direction it is pointing.

22.4.2 Constructor & Destructor Documentation

22.4.2.1 `Triton::Impact::Impact ( Ocean * pOcean, double pImpactorDiameter, double pMass, bool pSprayEffects = false, double pSprayScale = 2.0 )`

Construct a [Impact](#) with the [Triton::Ocean](#) it will be associated with.

Parameters

<i>pOcean</i>	The Triton::Ocean object you will associate this Impact with. A common error is to create an Impact before the ocean has been initialized, so make sure this is a valid, non-NULL ocean pointer.
<i>pImpactorDiameter</i>	The diameter of the disturbance in world units.
<i>pMass</i>	The mass of the impactor in grams
<i>pSprayEffects</i>	Whether you wish this impact to emit spray particles.
<i>pSprayScale</i>	A scaling factor applied to the initial velocity of spray particles.

22.4.3 Member Function Documentation

22.4.3.1 `Vector3 TRITONAPI Triton::Impact::GetPosition ( ) const [inline]`

Retrieves the world position of the [RotorWash](#).

Returns

The world position of the [RotorWash](#), as last specified by [Triton::RotorWash::Update\(\)](#).

22.4.3.2 `double TRITONAPI Triton::Impact::GetVelocity ( ) const [inline]`

Retrieves the velocity of the [RotorWash](#).

Returns

The velocity of the [RotorWash](#) in world units per second, as last specified by [Triton::RotorWash::Update\(\)](#).

22.4.3.3 `void TRITONAPI Triton::Impact::Trigger ( const Vector3 & pPosition, const Vector3 & pDirection, double pVelocity, double pTime )`

Starts the effect of an impact at a given position, direction, and velocity.

No impact will be generated until this is called.

Parameters

<i>pPosition</i>	The position of the rotor, in world coordinates.
------------------	--

<i>pDirection</i>	A normalized direction vector indicating the direction the rotors are pointing down toward.
<i>pVelocity</i>	The velocity of the object producing the wake, in meters per second.
<i>pTime</i>	The current simulated time sample, in seconds. This may be relative to any reference point in time, as long as that reference point is consistent among the multiple calls to Update().

The documentation for this class was generated from the following file:

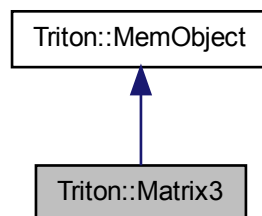
- C:/triton/trunk/Public Headers/[Impact.h](#)

22.5 Triton::Matrix3 Class Reference

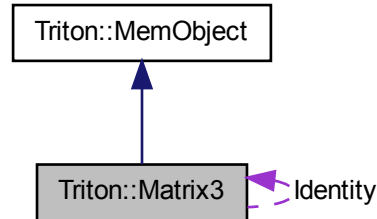
A simple 3x3 matrix class and its operations.

```
#include <Matrix3.h>
```

Inheritance diagram for Triton::Matrix3:



Collaboration diagram for Triton::Matrix3:



Public Member Functions

- [Matrix3](#) ()
Default contructor; performs no initialization for efficiency.
- [Matrix3](#) (double e11, double e12, double e13, double e21, double e22, double e23, double e31, double e32, double e33)
Constructor that instantiates the 3x3 matrix with initial values.
- [Matrix3](#) (double *m)
Constructor that takes an array of 9 doubles in row-major order.
- [~Matrix3](#) ()
Destructor.
- float *TRITONAPI [ToFloatArray](#) ()
Returns a static 3x3 float array in row major order.
- void TRITONAPI [FromRx](#) (double rad)
Populates the matrix to model a rotation about the X axis by a given amount, in radians.
- void TRITONAPI [FromRy](#) (double rad)
Populates the matrix to model a rotation about the Y axis by a given amount, in radians.
- void TRITONAPI [FromRz](#) (double rad)
Populates the matrix to model a rotation about the Z axis by a give amount, in radians.
- void TRITONAPI [FromXYZ](#) (double Rx, double Ry, double Rz)

Populates the matrix as a series of rotations about the X, Y, and Z axes (in that order) by specified amounts in radians.

- **Matrix3** TRITONAPI **operator*** (const **Matrix3** &mat)
Multiplies two matrices together.
- **Vector3** TRITONAPI **operator*** (const **Vector3** &rkVector) const
Multiplies the matrix by a vector, yielding another 3x1 vector.
- **Matrix3** TRITONAPI **Transpose** () const
Calculate the inverse of the matrix.

Public Attributes

- double **elem** [3][3]
The data members are public for convenience.

Friends

- **Vector3** TRITONAPI **operator*** (const **Vector3** &vec, const **Matrix3** &mat)
Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

22.5.1 Detailed Description

A simple 3x3 matrix class and its operations.

22.5.2 Constructor & Destructor Documentation

22.5.2.1 Triton::Matrix3::Matrix3 () [inline]

Default constructor; performs no initialization for efficiency.

22.5.2.2 Triton::Matrix3::Matrix3 (double e11, double e12, double e13, double e21, double e22, double e23, double e31, double e32, double e33) [inline]

Constructor that instantiates the 3x3 matrix with initial values.

22.5.2.3 Triton::Matrix3::Matrix3 (double * m) [inline]

Constructor that takes an array of 9 doubles in row-major order.

22.5.2.4 `Triton::Matrix3::~~Matrix3 ( ) [inline]`

Destructor.

22.5.3 Member Function Documentation**22.5.3.1** `void TRITONAPI Triton::Matrix3::FromRx ( double rad )`

Populates the matrix to model a rotation about the X axis by a given amount, in radians.

22.5.3.2 `void TRITONAPI Triton::Matrix3::FromRy ( double rad )`

Populates the matrix to model a rotation about the Y axis by a given amount, in radians.

22.5.3.3 `void TRITONAPI Triton::Matrix3::FromRz ( double rad )`

Populates the matrix to model a rotation about the Z axis by a give amount, in radians.

22.5.3.4 `void TRITONAPI Triton::Matrix3::FromXYZ ( double Rx, double Ry, double Rz )`

Populates the matrix as a series of rotations about the X, Y, and Z axes (in that order) by specified amounts in radians.

22.5.3.5 `Vector3 TRITONAPI Triton::Matrix3::operator* ( const Vector3 & rkVector ) const`

Multiplies the matrix by a vector, yielding another 3x1 vector.

22.5.3.6 `Matrix3 TRITONAPI Triton::Matrix3::operator* ( const Matrix3 & mat )`

Multiplies two matrices together.

22.5.3.7 `float* TRITONAPI Triton::Matrix3::ToFloatArray ( ) [inline]`

Returns a static 3x3 float array in row major order.

22.5.3.8 `Matrix3 TRITONAPI Triton::Matrix3::Transpose ( ) const`

Caculate the inverse of the matrix.

22.5.4 Friends And Related Function Documentation

22.5.4.1 Vector3 TRITONAPI operator* (const Vector3 & *vec*, const Matrix3 & *mat*) [friend]

Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

The documentation for this class was generated from the following file:

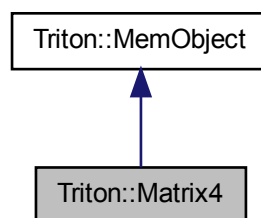
- C:/triton/trunk/Public Headers/[Matrix3.h](#)

22.6 Triton::Matrix4 Class Reference

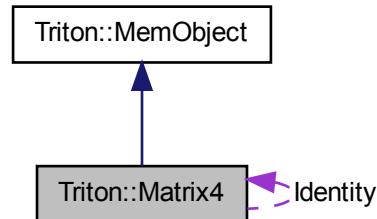
An implementation of a 4x4 matrix and some simple operations on it.

```
#include <Matrix4.h>
```

Inheritance diagram for Triton::Matrix4:



Collaboration diagram for Triton::Matrix4:



Public Member Functions

- [Matrix4](#) ()
Default constructor; initializes the matrix to an identity transform.
- [Matrix4](#) (double e11, double e12, double e13, double e14, double e21, double e22, double e23, double e24, double e31, double e32, double e33, double e34, double e41, double e42, double e43, double e44)
This constructor allows you to initialize the matrix as you please.
- [Matrix4](#) (const double *m)
Initializes the matrix from an array of 16 double-precision values (row-major).
- [~Matrix4](#) ()
Destructor.
- double TRITONAPI [operator\(\)](#) (int x, int y) const
Retrieve a specific matrix element.
- float *TRITONAPI [ToFloatArray](#) () const
Populates a static array of 16 floats with the contents of the matrix.
- [Matrix4](#) TRITONAPI [operator*](#) (const [Matrix4](#) &mat) const
Multiplies two matrices together.
- [Vector4](#) TRITONAPI [operator*](#) (const [Vector4](#) &vec) const
Transform a point by the matrix.
- [Vector3](#) TRITONAPI [operator*](#) (const [Vector3](#) &vec) const

Transform a point by the matrix.

- void TRITONAPI [Transpose](#) ()
Transposes the matrix in-place.
- [Matrix4](#) TRITONAPI [InverseCramers](#) (double epsilon=1E-9)
Computes the inverse of the matrix using Cramer's rule.
- double *TRITONAPI [GetRow](#) (int row) const
Retrieves a pointer into the requested row of the matrix.

Public Attributes

- double [elem](#) [4][4]
Data members are public for convenience.

Friends

- [Vector4](#) TRITONAPI [operator*](#) (const [Vector4](#) &vec, const [Matrix4](#) &mat)
Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

22.6.1 Detailed Description

An implementation of a 4x4 matrix and some simple operations on it.

22.6.2 Constructor & Destructor Documentation

22.6.2.1 Triton::Matrix4::Matrix4 () [inline]

Default constructor; initializes the matrix to an identity transform.

22.6.2.2 Triton::Matrix4::Matrix4 (double *e11*, double *e12*, double *e13*, double *e14*, double *e21*, double *e22*, double *e23*, double *e24*, double *e31*, double *e32*, double *e33*, double *e34*, double *e41*, double *e42*, double *e43*, double *e44*) [inline]

This constructor allows you to initialize the matrix as you please.

22.6.2.3 Triton::Matrix4::Matrix4 (const double * *m*) [inline]

Initializes the matrix from an array of 16 double-precision values (row-major).

22.6.2.4 `Triton::Matrix4::~~Matrix4 ( ) [inline]`

Destructor.

22.6.3 Member Function Documentation**22.6.3.1** `double* TRITONAPI Triton::Matrix4::GetRow ( int row ) const [inline]`

Retrieves a pointer into the requested row of the matrix.

22.6.3.2 `Matrix4 TRITONAPI Triton::Matrix4::InverseCramers ( double epsilon = 1E-9 ) [inline]`

Computes the inverse of the matrix using Cramer's rule.

22.6.3.3 `Vector4 TRITONAPI Triton::Matrix4::operator* ( const Vector4 & vec ) const`

Transform a point by the matrix.

22.6.3.4 `Vector3 TRITONAPI Triton::Matrix4::operator* ( const Vector3 & vec ) const`

Transform a point by the matrix.

22.6.3.5 `Matrix4 TRITONAPI Triton::Matrix4::operator* ( const Matrix4 & mat ) const`

Multiplies two matrices together.

22.6.3.6 `float* TRITONAPI Triton::Matrix4::ToFloatArray ( ) const [inline]`

Populates a static array of 16 floats with the contents of the matrix.

22.6.3.7 `void TRITONAPI Triton::Matrix4::Transpose ( ) [inline]`

Transposes the matrix in-place.

22.6.4 Friends And Related Function Documentation**22.6.4.1** `Vector4 TRITONAPI operator* ( const Vector4 & vec, const Matrix4 & mat ) [friend]`

Multiplies a 1x3 vector by a matrix, yielding a 1x3 vector.

The documentation for this class was generated from the following file:

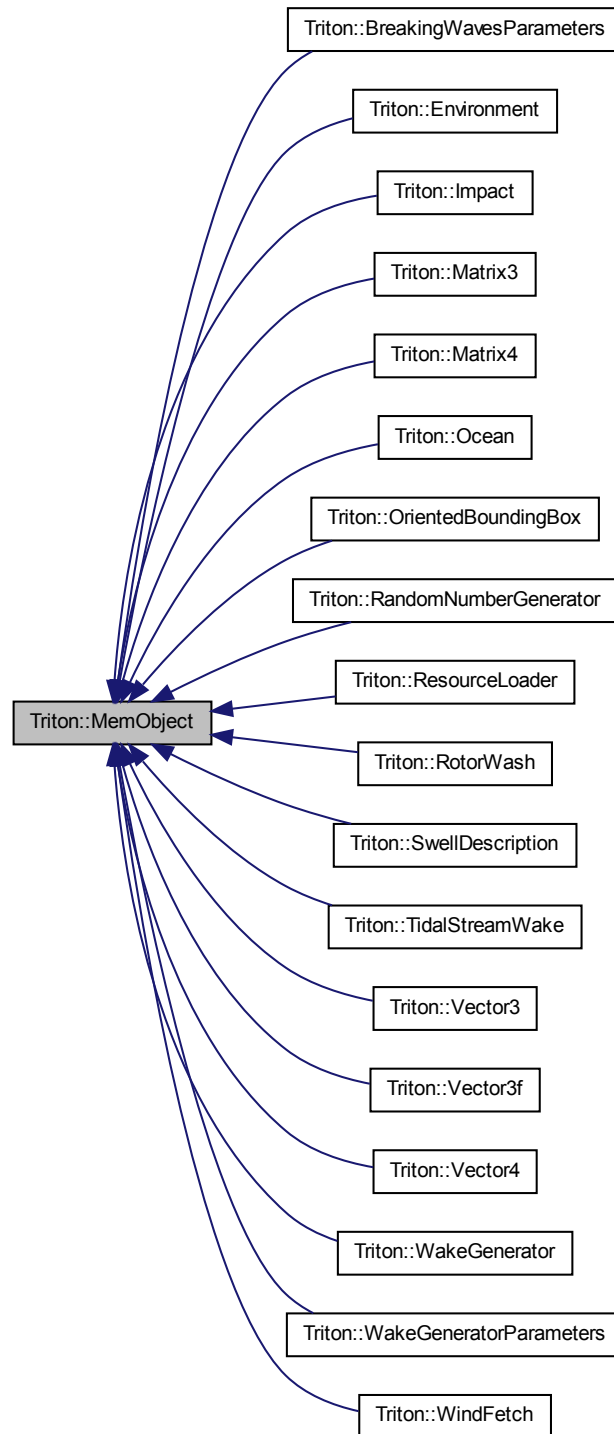
- C:/triton/trunk/Public Headers/[Matrix4.h](#)

22.7 Triton::MemObject Class Reference

This base class for all Triton objects intercepts the new and delete operators, routing them through Triton::Allocator().

```
#include <MemAlloc.h>
```

Inheritance diagram for Triton::MemObject:



22.7.1 Detailed Description

This base class for all Triton objects intercepts the new and delete operators, routing them through Triton::Allocator().

The documentation for this class was generated from the following file:

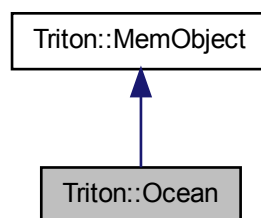
- C:/triton/trunk/Public Headers/[MemAlloc.h](#)

22.8 Triton::Ocean Class Reference

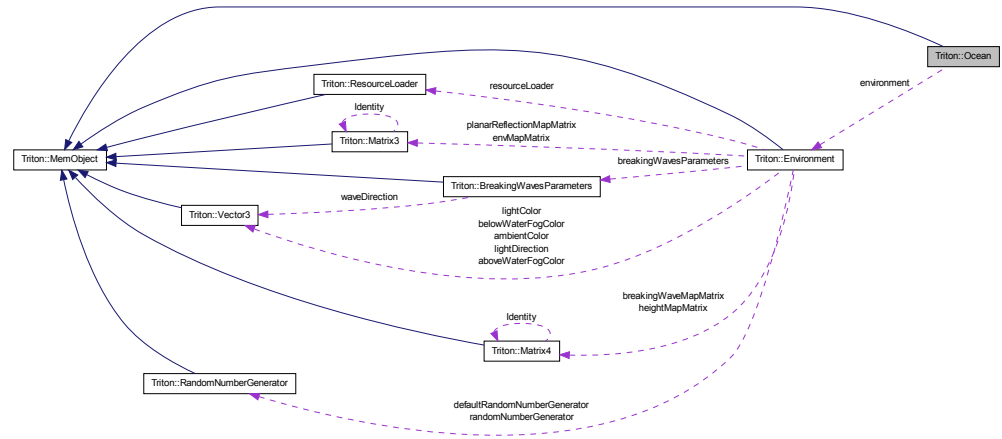
The [Ocean](#) class allows you to configure and draw Triton's water simulation.

```
#include <Ocean.h>
```

Inheritance diagram for Triton::Ocean:



Collaboration diagram for Triton::Ocean:



Public Member Functions

- virtual `~Ocean()`
Virtual destructor.
- virtual void TRITONAPI `Draw` (double time, bool depthWrites=true, bool drawWater=true, bool drawParticles=true)
Draws an infinite ocean surrounding the camera (as specified in the [Environment](#) object) for the simulated conditions at the given time.
- virtual bool TRITONAPI `SetPatchShader` (double time, int vertexStride, int positionOffset, bool doublePrecisionVertices, const double *modelMatrix=0)
Sets the shaders and state necessary for rendering a user-provided patch of geometry as water.
- virtual bool TRITONAPI `SetPatchMatrix` (const double *modelMatrix)
If you are drawing many of your own water meshes using `SetPatchShader()` at once, it will be much faster to call `SetPatchShader()` once, then call `SetPatchMatrix()` for each individual mesh to set its location, followed by `UnsetPatchShader()` when you're done drawing all of them.
- virtual void TRITONAPI `UnsetPatchShader` (double time=0.0f, const TBoundingBox *patchBounds=0)
Restores the graphics state prior to a previous call to `Ocean::SetPatchShader()`.
- virtual bool `ReloadShaders` (const TRITON_VECTOR(unsigned int)&shaders)

OpenGL only: Reload the underlying shader programs, linking in a new list of user-supplied shader object ID's with each program.

- virtual void TRITONAPI [UpdateSimulation](#) (double time)
Updates the underlying wave simulation; calling this is optional and only necessary if you wish to perform physics updates from a pass or thread different from the rendering pass or thread.
- void TRITONAPI [D3D9DeviceLost](#) ()
DirectX9 users must call [D3D9DeviceLost\(\)](#) in response to lost devices, prior to re-setting the device.
- void TRITONAPI [D3D9DeviceReset](#) ()
DirectX9 users must call [D3D9DeviceReset\(\)](#) in response to device resets done in response to lost devices.
- float TRITONAPI [GetChoppiness](#) () const
Retrieves the choppiness setting of the [Ocean](#), which controls how peaked the waves are.
- void TRITONAPI [SetChoppiness](#) (float chop)
Set the choppiness of the waves, which controls how peaked the waves are.
- float TRITONAPI [GetLoopingPeriod](#) () const
Retrieves the looping period of the [Ocean](#), which define time after which wave simulation repeats.
- void TRITONAPI [SetLoopingPeriod](#) (float loopingPeriod)
Set the looping period of the [Ocean](#), which define time after which wave simulation repeats.
- float TRITONAPI [GetDepth](#) ([Triton::Vector3](#) &floorNormal) const
Retrieves the simulated depth of the water in world units, and the surface normal of the sea floor, both at the current camera position.
- void TRITONAPI [SetDepth](#) (float depth, const [Triton::Vector3](#) &floorNormal)
Sets the simulated depth of the water in world units at the camera position, and the slope of the sea floor as specified by its surface normal at the camera position.
- void TRITONAPI [EnableWireframe](#) (bool wireframeOn)
Enables or disables wireframe rendering of the ocean's mesh.
- const char *TRITONAPI [GetFFTName](#) () const
Returns a description of the FFT transform being used, if a FFT water model is active.
- unsigned int TRITONAPI [GetNumTriangles](#) () const
Returns the number of triangles rendered by the underlying projected grid.

- ShaderHandle TRITONAPI [GetShaderObject](#) (Shaders shaderProgram) const
Retrieves an underlying shader object used to render the water.
- DecalHandle TRITONAPI [AddDecal](#) (TextureHandle texture, float size, const [Vector3](#) &position)
Applies a decal texture to the dynamic ocean surface, useful for effects involving films, debris, or foam on the water.
- void TRITONAPI [ScaleDecal](#) (DecalHandle decal, float scaleWidth, float scaleDepth)
Scales an existing decal in width and depth at runtime.
- void TRITONAPI [SetDecalAlpha](#) (DecalHandle decal, float alpha)
Sets a decal's alpha blending amount (default is 1.0.)
- void TRITONAPI [RemoveDecal](#) (DecalHandle decal)
Removes a decal texture previously applied with [AddDecal\(\)](#).
- bool TRITONAPI [GetHeight](#) (const [Vector3](#) &point, const [Vector3](#) &direction, float &height, [Vector3](#) &normal, bool visualCorrelation=true, bool includeWakes=true, bool highResolution=true, bool threadSafe=true)
Retrieves the height and normal of the ocean surface at the intersection point of the given ray.
- bool TRITONAPI [GetIntersection](#) (const [Vector3](#) &point, const [Vector3](#) &direction, [Vector3](#) &intersection)
Retrieves the intersection, if any, between a ray and the ocean surface.
- float TRITONAPI [GetWaveHeading](#) () const
Retrieves the wave direction.
- void TRITONAPI [EnableSpray](#) (bool enable)
Enables or disables spray particle effects on breaking waves.
- bool TRITONAPI [SprayEnabled](#) () const
Returns if spray particle effects on breaking waves are enabled, which they are by default.
- void TRITONAPI [SetRefractionColor](#) (const [Vector3](#) &refractionColor)
Modifies the color used for refracted light rays that go into deep water.
- const [Vector3](#) &TRITONAPI [GetRefractionColor](#) () const
Returns the color of light refracted into the water.
- void TRITONAPI [SetPlanarReflectionBlend](#) (float blendPercent)
Sets the prominence of planar reflections on this ocean surface, if one was set using [Triton::Environment::SetPlanarReflectionMap\(\)](#).

- float TRITONAPI [GetPlanarReflectionBlend](#) () const
Retrieves the current blend percentage for planar reflections.
- bool TRITONAPI [ComputeReflectionMatrices](#) (Matrix4 &reflectionMatrix, Matrix3 &textureMatrix)
A helper function for using planar reflections with Triton.
- const [Environment](#) *TRITONAPI [GetEnvironment](#) () const
Retrieves the environment this ocean is attached to.
- bool TRITONAPI [IsCameraAboveWater](#) ()
Returns whether the current camera position (from [Environment::SetCameraMatrix\(\)](#)) is above the simulated water surface for this [Ocean](#).
- void TRITONAPI [SetDepthOffset](#) (float offset)
Applies a depth offset to the water, applied in the vertex program.
- float TRITONAPI [GetDepthOffset](#) () const
Retrieves the depth offset (if any) previously set via [Triton::Ocean::SetDepthOffset\(\)](#), used to combat "z fighting" near coastlines.
- void TRITONAPI [SetDisplacementDampingDistance](#) (double distance)
Sets the distance at which 3D wave displacements are dampened to prevent aliasing when moving the camera.
- double TRITONAPI [GetDisplacementDampingDistance](#) () const
Retrieves the distance at which 3D wave displacements are dampened to prevent aliasing when moving the camera.
- void TRITONAPI [EnableGodRays](#) (bool enable)
Turns the underwater crepuscular rays effect on or off.
- bool TRITONAPI [GodRaysEnabled](#) () const
Returns whether the underwater crepuscular rays effect is enabled.
- void TRITONAPI [SetGodRaysFade](#) (float fadeAmount)
Fades out the underwater crepuscular rays effect by the specified amount (0 = no fading, 1 = completely faded).
- float TRITONAPI [GetGodRaysFade](#) () const
Returns the god ray fading amount set in [Ocean::FadeGodRays\(\)](#).
- void TRITONAPI [Lock](#) ()
Explicitly locks the mutex used to ensure thread safety between the draw, update, and height query methods.

- void TRITONAPI [Unlock](#) ()
Explicitly locks the ocean's mutex previously locked by [Ocean::Lock\(\)](#).
- void TRITONAPI [SetQuality](#) (OceanQuality quality)
Set a quality setting (GOOD, BETTER, or BEST.) Higher quality will result in finer wave resolution, but at lower performance.
- OceanQuality TRITONAPI [GetQuality](#) () const
Retrieve the current simulation quality setting, either set by [Ocean::SetQuality\(\)](#) or the default value of GOOD.
- void TRITONAPI [SetLinearColorSpace](#) (bool linearOn)
Sets use of linear color space, in which the ocean color will be raised to the power of 2.2 to negate the effects of gamma correction.
- bool TRITONAPI [GetLinearColorSpace](#) () const
Gets whether linear color space rendering is enabled via [SetLinearColorSpace\(\)](#).

Static Public Member Functions

- static [Ocean](#) *TRITONAPI [Create](#) ([Environment](#) *env, WaterModelTypes type=JONSWAP, bool enableHeightTests=false, bool enableBreakingWaves=false, OceanQuality quality=GOOD)
Creates an [Ocean](#) instance tied to the given [Environment](#), using the specified wave model.
- static [Ocean](#) *TRITONAPI [Create](#) ([Environment](#) *env, const TRITON_VECTOR(unsigned int)&userShaders, WaterModelTypes type=JONSWAP, bool enableHeightTests=false, bool enableBreakingWaves=false, OceanQuality quality=GOOD)
Creates an [Ocean](#) instance tied to the given [Environment](#), using additional user-supplied OpenGL shader objects.

22.8.1 Detailed Description

The [Ocean](#) class allows you to configure and draw Triton's water simulation.

22.8.2 Constructor & Destructor Documentation

22.8.2.1 virtual Triton::Ocean::~~Ocean () [virtual]

Virtual destructor.

22.8.3 Member Function Documentation

22.8.3.1 DecalHandle TRITONAPI Triton::Ocean::AddDecal (TextureHandle *texture*, float *size*, const Vector3 & *position*)

Applies a decal texture to the dynamic ocean surface, useful for effects involving films, debris, or foam on the water.

Parameters

<i>texture</i>	The texture to apply to the water surface. Under OpenGL, this must be a GLuint indicating the ID of the GL_TEXTURE_2D returned from glGenTextures. Under DirectX9, this must be a LPDIRECT3DTEXTURE9. Under DirectX11, this must be a ID3D11ShaderResourceView pointer with an underlying ViewDimension of D3D11_SRV_DIMENSION_TEXTURE2D.
<i>size</i>	The size of the decal, in world units.
<i>position</i>	The center position of the decal on the water surface, in world units.

Returns

An identifier for this decal, which may be passed to [RemoveDecal\(\)](#) to later dispose of this decal.

22.8.3.2 bool TRITONAPI Triton::Ocean::ComputeReflectionMatrices (Matrix4 & *reflectionMatrix*, Matrix3 & *textureMatrix*)

A helper function for using planar reflections with Triton.

This will suggest useful matrices for flipping your scene about the plane of the local water surface, and a texture matrix to transform world vectors into texture coordinates in a reflection map. You may need to transpose and/or adjust these matrices for left handed coordinates, depending on the conventions of your engine. The texture matrix returned will take the differences between OpenGL and DirectX normalized device coordinates and texture coordinates into account, however.

See also

[Triton::Environment::SetPlanarReflectionMap\(\)](#)

Parameters

<i>reflection-Matrix</i>	A matrix that will flip the scene about the local water plane, taking the current sea level into account. If you multiply this matrix into your modelview matrix after your camera transforms, all objects will be flipped about the water plane - making them suitable for rendering into a reflection texture map to later be passed in to Triton::Environment::SetPlanarReflectionMap() . You will need to change the winding order used for back face culling when this matrix is active, and you may also need to enable a user clipping plane to prevent geometry that is normally underwater from rendering into your reflection texture.
--------------------------	--

<i>textureMatrix</i>	A matrix to transform world vectors into texture coordinate space of the reflection map texture. This matrix would be passed into Triton::Environment::SetPlanarReflectionMap() . This is equal to the product of view * projection * NDCtoTexCoords, based on the view and projection matrices last passed to Environment::SetCameraMatrix() and Environment::SetProjectionMatrix() . The matrix to transform normalized device coordinates to texture coordinates will differ depending on whether an OpenGL or DirectX renderer is being used.
----------------------	---

Returns

True if valid matrices were computed; will return false if the camera is below sea level, in which case you probably don't want reflections anyhow.

22.8.3.3 `static Ocean* TRITONAPI Triton::Ocean::Create ( Environment * env, WaterModelTypes type = JONSWAP, bool enableHeightTests = false, bool enableBreakingWaves = false, OceanQuality quality = GOOD ) [static]`

Creates an [Ocean](#) instance tied to the given [Environment](#), using the specified wave model.

Make sure this is called only after your graphics context has been initialized and is active!

Parameters

<i>env</i>	A pointer to an Environment object created previously, which contains the environmental conditions, coordinate system, camera, and rendering system used by the Ocean . The caller is responsible for deleting this object after the Ocean is deleted.
<i>type</i>	You may choose between the faster TESSENDORF implementation, or the more physically accurate JONSWAP and PIERSON_MOSKOWITZ spectral models. JONSWAP is an extension of PIERSON_MOSKOWITZ that accounts for wind "fetch length," so be sure to specify realistic distances (~100km) travelled by the wind in your WindFetch objects to make the most of JONSWAP. The GERSTNER model is deprecated.
<i>enableHeightTests</i>	Specifies whether the application will call Ocean::GetHeight() or not. If false, Triton may be able to keep the ocean simulation entirely on the GPU leading to better performance, but any calls to Ocean::GetHeight() may return 0. Set to true if you need to read back height information from the water surface.
<i>enableBreakingWaves</i>	Specifies whether shoreline effects with breaking waves are enabled. This does create additional demands on the vertex and fragment programs and should only be enabled if you're going to use them. Breaking waves require a floating-point height map containing bathymetry data to be passed in via Environment::SetHeightMap() .

See also

[Environment::SetBreakingWavesParameters\(\)](#)

Parameters

<i>quality</i>	Specifies the tradeoff you want between wave detail and performance. Choose from GOOD, BETTER, or BEST.
----------------	---

Returns

An instance of [Ocean](#) that may be used for rendering. The caller is responsible for deleting this object when finished. NULL may be returned if the ocean could not initialize itself; in this case, enable the setting enable-debug-messages in resource/triton.config to get more details on what went wrong sent to your debugger output window. Contact support@sundog-soft.com with this output if necessary.

```
22.8.3.4 static Ocean* TRITONAPI Triton::Ocean::Create ( Environment * env, const
TRITON_VECTOR(unsigned int)& userShaders, WaterModelTypes type = JONSWAP,
bool enableHeightTests = false, bool enableBreakingWaves = false,
OceanQuality quality = GOOD ) [static]
```

Creates an [Ocean](#) instance tied to the given [Environment](#), using additional user-supplied OpenGL shader objects.

Make sure this is called only after your graphics context has been initialized and is active!

Parameters

<i>env</i>	A pointer to an Environment object created previously, which contains the environmental conditions, coordinate system, camera, and rendering system used by the Ocean . The caller is responsible for deleting this object after the Ocean is deleted.
<i>userShaders</i>	A vector of OpenGL shader objects (created via glCreateShader) that will be linked into all shader programs created by Triton. This allows you to provide your own shader functions to Triton, which your modified Triton shaders may then call into. Triton::Ocean::GetShaderObject() may be used to retrieve the linked shader programs, allowing you to set any uniforms your functions require. The caller is responsible for deleting these custom shader object after this Ocean object has been deleted.
<i>type</i>	You may choose between the faster TESSENDORF implementation, or the more physically accurate JONSWAP and PIERSON_MOSKOWITZ spectral models. JONSWAP is an extension of PIERSON_MOSKOWITZ that accounts for wind "fetch length," so be sure to specify realistic distances (~100km) travelled by the wind in your WindFetch objects to make the most of JONSWAP. The GERSTNER model is deprecated.

<i>enable-HeightTests</i>	Specifies whether the application will call Ocean::GetHeight() or not. If false, Triton may be able to keep the ocean simulation entirely on the GPU leading to better performance, but any calls to Ocean::GetHeight() may return 0. Set to true if you need to read back height information from the water surface.
<i>enable-Breaking-Waves</i>	Specifies whether shoreline effects with breaking waves are enabled. This does create additional demands on the vertex and fragment programs and should only be enabled if you're going to use them. Breaking waves require a floating-point height map containing bathymetry data to be passed in via Environment::SetHeightMap() .

See also

[Environment::SetBreakingWavesParameters\(\)](#)

Parameters

<i>quality</i>	Specifies the tradeoff you want between wave detail and performance. Choose from GOOD, BETTER, or BEST.
----------------	---

Returns

An instance of [Ocean](#) that may be used for rendering. The caller is responsible for deleting this object when finished. NULL may be returned if the ocean could not initialize itself; in this case, enable the setting enable-debug-messages in resource/triton.config to get more details on what went wrong sent to your debugger output window. Contact support@sundog-soft.com with this output if necessary.

22.8.3.5 void TRITONAPI Triton::Ocean::D3D9DeviceLost ()

DirectX9 users must call [D3D9DeviceLost\(\)](#) in response to lost devices, prior to resetting the device.

A lost device may occur when the user locks and unlocks a system or changes the monitor resolution, and must be explicitly handled under DX9. Call [D3D9DeviceReset\(\)](#) once the device has been recreated.

22.8.3.6 void TRITONAPI Triton::Ocean::D3D9DeviceReset ()

DirectX9 users must call [D3D9DeviceReset\(\)](#) in response to device resets done in response to lost devices.

You must have called [D3D9DeviceLost\(\)](#) first in response to the lost device.

22.8.3.7 `virtual void TRITONAPI Triton::Ocean::Draw ( double time, bool depthWrites = true, bool drawWater = true, bool drawParticles = true ) [virtual]`

Draws an infinite ocean surrounding the camera (as specified in the [Environment](#) object) for the simulated conditions at the given time.

Parameters

<i>time</i>	The simulated point in time to render, in seconds. Note that this is an absolute time which can be relative to any arbitrary point in time. It's not the delta time between frames.
<i>depthWrites</i>	Whether the ocean will write to the depth buffer; you should probably leave this set to true, since disabling it will lead to some artifacts. One technique for integrating the ocean with terrain is to draw the ocean first with depth writes disabled, then draw the terrain on top of it, thereby avoiding any depth buffer precision issues. Be sure to remove or disable any existing ocean surfaces from your terrain database first if using this technique. Be aware that disabling depth writes will lead to artifacts at certain camera angles, since the waves won't be able to depth-sort against themselves; a better and simpler approach would be to leave depthWrites on, and then clear the depth buffer after calling Draw early in your frame.
<i>drawWater</i>	Whether the water surface should be drawn in this call. You may want to draw the water (which is non-transparent) separately from the spray particles (which are transparent.) This parameter, together with drawParticles, lets you choose what combination of water and particles should be drawn with this call.
<i>drawParticles</i>	Whether spray particles from breaking waves, rotor wash, impacts, etc. are drawn with this call.

22.8.3.8 `void TRITONAPI Triton::Ocean::EnableGodRays ( bool enable ) [inline]`

Turns the underwater crepuscular rays effect on or off.

If the Triton.config setting underwater-god-rays-enabled is set to "no", this will have no effect. Defaults to false.

22.8.3.9 `void TRITONAPI Triton::Ocean::EnableSpray ( bool enable )`

Enables or disables spray particle effects on breaking waves.

This does incur a performance penalty, so if you need faster performance, try disabling spray effects or even disable it entirely with the fft-enable-spray config setting in resources/Triton.config.

22.8.3.10 `void TRITONAPI Triton::Ocean::EnableWireframe ( bool wireframeOn )`

Enables or disables wireframe rendering of the ocean's mesh.

Parameters

<i>wire-frameOn</i>	Set to true to render in wireframe mode, false to render normally.
---------------------	--

22.8.3.11 float TRITONAPI Triton::Ocean::GetChoppiness () const

Retrieves the choppiness setting of the [Ocean](#), which controls how peaked the waves are.

See also

[SetChoppiness\(\)](#)

22.8.3.12 float TRITONAPI Triton::Ocean::GetDepth (Triton::Vector3 & floorNormal) const

Retrieves the simulated depth of the water in world units, and the surface normal of the sea floor, both at the current camera position.

Only returns the values set by [SetDepth\(\)](#).

See also

[SetDepth\(\)](#)

Parameters

<i>floorNormal</i>	A reference to a Vector3 to retrieve the surface normal of the sea floor as set by SetDepth() .
--------------------	---

Returns

The depth of the sea floor at the camera position, as set by [SetDepth\(\)](#).

22.8.3.13 float TRITONAPI Triton::Ocean::GetDepthOffset () const

Retrieves the depth offset (if any) previously set via [Triton::Ocean::SetDepthOffset\(\)](#), used to combat "z fighting" near coastlines.

22.8.3.14 double TRITONAPI Triton::Ocean::GetDisplacementDampingDistance () const

Retrieves the distance at which 3D wave displacements are dampened to prevent aliasing when moving the camera.

22.8.3.15 const Environment* TRITONAPI Triton::Ocean::GetEnvironment () const
[inline]

Retrieves the environment this ocean is attached to.

22.8.3.16 `const char* TRITONAPI Triton::Ocean::GetFFTName ( ) const`

Returns a description of the FFT transform being used, if a FFT water model is active.

22.8.3.17 `float TRITONAPI Triton::Ocean::GetGodRaysFade ( ) const`

Returns the god ray fading amount set in `Ocean::FadeGodRays()`.

22.8.3.18 `bool TRITONAPI Triton::Ocean::GetHeight ( const Vector3 & point, const Vector3 & direction, float & height, Vector3 & normal, bool visualCorrelation = true, bool includeWakes = true, bool highResolution = true, bool threadSafe = true )`

Retrieves the height and normal of the ocean surface at the intersection point of the given ray.

The results of this method are only valid if `Ocean::Create()` was called with the parameter `enableHeightReads` set to true. The height returned is relative to sea level, as specified by `Triton::Environment::SetSeaLevel()`. For example, the crest of a one-meter-high wave will always return a height of one meter, irrespective of the environment's sea level height. Depending on the application, you may want to add in the result of `Triton::Environment::GetSeaLevel()` to the height returned.

Parameters

<i>point</i>	The origin of the ray to test the height against.
<i>direction</i>	The normalized direction vector of the ray.
<i>height</i>	Receives the height at the ray's intersection with the ocean, if an intersection was found.
<i>normal</i>	Receives a normalized unit vector pointing in the direction of the normal vector of the sea surface at the intersection point.
<i>visualCorrelation</i>	Set to true in order to have the height returned match the visuals, which dampen height offsets with distance to avoid sampling artifacts. To return the true wave height at the given location, set to false.
<i>includeWakes</i>	Whether this height query should include waves from rotor wash and impacts in its results. Ship Kelvin wakes won't be included.
<i>highResolution</i>	If true, the ocean grid points surrounding the intersection point will be sampled and interpolated. If false, the height will be based on the nearest grid point alone.
<i>threadSafe</i>	Whether a mutex will be locked with this call to ensure thread safety with <code>Ocean::Draw()</code> . If you are conducting many intersection tests and are only single threaded, pass false for better performance. You may also use <code>Ocean::Lock()</code> and <code>Ocean::Unlock()</code> to manually lock and unlock the mutex surrounding many <code>GetHeight</code> calls.

Returns

True if an intersection was found, false if not. If a height map was passed in using `Environment::SetHeightMap()`, this will return false if the intersection is over terrain.

22.8.3.19 `bool TRITONAPI Triton::Ocean::GetIntersection ( const Vector3 & point, const Vector3 & direction, Vector3 & intersection )`

Retrieves the intersection, if any, between a ray and the ocean surface.

This method does NOT take wave heights into account; use [Ocean::GetHeight\(\)](#) for that level of resolution.

Parameters

<i>point</i>	The origin of the ray to intersect with, in world coordinates.
<i>direction</i>	The direction of the ray to intersect with, in world coordinates.
<i>intersection</i>	Receives the intersection point if an intersection exists.

Returns

True if an intersection was found, false if not. If height map data has been given using [Environment::SetHeightMap\(\)](#), false will also be returned if the intersection point is over terrain.

22.8.3.20 `bool TRITONAPI Triton::Ocean::GetLinearColorSpace ( ) const`

Gets whether linear color space rendering is enabled via [SetLinearColorSpace\(\)](#).

22.8.3.21 `float TRITONAPI Triton::Ocean::GetLoopingPeriod ( ) const`

Retrieves the looping period of the [Ocean](#), which define time after which wave simulation repeats.

See also

[SetLoopingPeriod\(\)](#)

22.8.3.22 `unsigned int TRITONAPI Triton::Ocean::GetNumTriangles ( ) const`

Returns the number of triangles rendered by the underlying projected grid.

22.8.3.23 `float TRITONAPI Triton::Ocean::GetPlanarReflectionBlend ( ) const`

Retrieves the current blend percentage for planar reflections.

See also

[SetPlanarReflectionBlend\(\)](#).

22.8.3.24 OceanQuality TRITONAPI Triton::Ocean::GetQuality () const [inline]

Retrieve the current simulation quality setting, either set by [Ocean::SetQuality\(\)](#) or the default value of GOOD.

22.8.3.25 const Vector3& TRITONAPI Triton::Ocean::GetRefractionColor () const

Returns the color of light refracted into the water.

See also

[SetRefractionColor\(\)](#);

Returns

The RGB value of the refraction color.

22.8.3.26 ShaderHandle TRITONAPI Triton::Ocean::GetShaderObject (Shaders shaderProgram) const

Retrieves an underlying shader object used to render the water.

If you make modifications to our shaders to add additional effects, you can use this in order to pass your own uniform variables into the shaders. Depending on the renderer you're using, you'll need to cast this to a GLhandleARB, ID3DXEffect, or ID3DX11Effect.

If you are shading the water surface, be sure to pass uniforms to both WATER_SURFACE and WATER_SURFACE_PATCH. The latter is used when the camera is near the water surface.

Parameters

<i>shaderProgram</i>	Specifies which shader program you want to retrieve: WATER_SURFACE, WATER_SURFACE_PATCH, SPRAY_PARTICLES, or WAKE_SPRAY_PARTICLES.
----------------------	--

Returns

The GLhandleARB, ID3DXEffect, or ID3DX11Effect representing the shader object, or 0 if no shader is loaded.

22.8.3.27 float TRITONAPI Triton::Ocean::GetWaveHeading () const [inline]

Retrieves the wave direction.

Normally this is the same as the wind direction, but in shallow water it will align with the slope of the sea floor as specified in [SetDepth\(\)](#).

22.8.3.28 `bool TRITONAPI Triton::Ocean::GodRaysEnabled ( ) const [inline]`

Returns whether the underwater crepuscular rays effect is enabled.

22.8.3.29 `bool TRITONAPI Triton::Ocean::IsCameraAboveWater ( )`

Returns whether the current camera position (from [Environment::SetCameraMatrix\(\)](#)) is above the simulated water surface for this [Ocean](#).

22.8.3.30 `void TRITONAPI Triton::Ocean::Lock ( )`

Explicitly locks the mutex used to ensure thread safety between the draw, update, and height query methods.

For example, if you wish to conduct many calls to [Ocean::GetHeight\(\)](#), it is more efficient to enclose them with calls to Lock / Unlock and then pass false for [Ocean::GetHeight\(\)](#)'s `threadSafety` parameter. Make sure this call is balanced by a call to [Triton::Unlock\(\)](#) under every circumstance.

22.8.3.31 `virtual bool Triton::Ocean::ReloadShaders ( const TRITON_VECTOR(unsigned int)& shaders ) [virtual]`

OpenGL only: Reload the underlying shader programs, linking in a new list of user-supplied shader object ID's with each program.

22.8.3.32 `void TRITONAPI Triton::Ocean::RemoveDecal ( DecalHandle decal )`

Removes a decal texture previously applied with [AddDecal\(\)](#).

22.8.3.33 `void TRITONAPI Triton::Ocean::ScaleDecal ( DecalHandle decal, float scaleWidth, float scaleDepth )`

Scales an existing decal in width and depth at runtime.

Parameters

<i>decal</i>	A decal handle retrieved from a previous call to AddDecal() .
<i>scaleWidth</i>	The scale factor for the decal's width (ie, 0.5 = half size, 2.0 = double.)
<i>scaleDepth</i>	The scale factor for the decal's depth.

22.8.3.34 `void TRITONAPI Triton::Ocean::SetChoppiness ( float chop )`

Set the choppiness of the waves, which controls how peaked the waves are.

See also

[GetChoppiness\(\)](#)

Parameters

<i>chop</i>	The choppiness parameter; 0.0 yields no chop, 3.0 yields strong chop. Values that are too high may result in wave geometry folding over itself, so take care to set reasonable values.
-------------	--

22.8.3.35 void TRITONAPI Triton::Ocean::SetDecalAlpha (DecalHandle *decal*, float *alpha*)

Sets a decal's alpha blending amount (default is 1.0.)

Parameters

<i>decal</i>	A decal handle retrieved from a previous call to AddDecal() .
<i>alpha</i>	The alpha value applied to the decal texture. 0 = transparent, 1.0 = opaque.

22.8.3.36 void TRITONAPI Triton::Ocean::SetDepth (float *depth*, const Triton::Vector3 & *floorNormal*)

Sets the simulated depth of the water in world units at the camera position, and the slope of the sea floor as specified by its surface normal at the camera position.

This information is used to interpolate the water depth at various positions in the scene, affecting the transparency of the water as well as the height of the waves. Avoid changing this every frame for performance reasons.

See also

[GetDepth\(\)](#)

Parameters

<i>depth</i>	The depth of the water, in world units, at the camera position. Negative values will be clamped to zero. For open ocean, either do not call this method or set depth to a large number (like 1000).
<i>floorNormal</i>	The surface normal of the sea floor at the camera position. The point defined by the depth parameter under the camera position together with this normal will define a plane that approximates the position of the sea floor surrounding the current location.

22.8.3.37 void TRITONAPI Triton::Ocean::SetDepthOffset (float *offset*)

Applies a depth offset to the water, applied in the vertex program.

This can be used to mitigate "z fighting" artifacts near shorelines. By default, there is no depth offset. A value of 0.01 is generally effective.

Parameters

<i>offset</i>	The depth offset subtracted from the Z (depth) value in normalized coordinate space when rendering the ocean.
---------------	---

22.8.3.38 void TRITONAPI Triton::Ocean::SetDisplacementDampingDistance (double *distance*)

Sets the distance at which 3D wave displacements are dampened to prevent aliasing when moving the camera.

22.8.3.39 void TRITONAPI Triton::Ocean::SetGodRaysFade (float *fadeAmount*)

Fades out the underwater crepuscular rays effect by the specified amount (0 = no fading, 1 = completely faded).

22.8.3.40 void TRITONAPI Triton::Ocean::SetLinearColorSpace (bool *linearOn*)

Sets use of linear color space, in which the ocean color will be raised to the power of 2.2 to negate the effects of gamma correction.

Only set this if you know you are rendering in linear color space.

22.8.3.41 void TRITONAPI Triton::Ocean::SetLoopingPeriod (float *loopingPeriod*)

Set the looping period of the [Ocean](#), which define time after which wave simulation repeats.

See also

[GetLoopingPeriod\(\)](#) If set to non zero, wave simulation loops after that period. 0 is the default. FFT Wave model uses mixture of compound frequency waves. When Period value is set, these compound frequencies are constrained to be a multiply of looping frequency. Hence period may affect the wave shapes. General rule is to use highest possible period length to minimize the impact on wave field realism. We recommend periods larger than 30 seconds.

22.8.3.42 virtual bool TRITONAPI Triton::Ocean::SetPatchMatrix (const double * *modelMatrix*)
[virtual]

If you are drawing many of your own water meshes using [SetPatchShader\(\)](#) at once, it will be much faster to call [SetPatchShader\(\)](#) once, then call [SetPatchMatrix\(\)](#) for each individual mesh to set its location, followed by [UnsetPatchShader\(\)](#) when you're done drawing all of them.

This is only useful if you are drawing many water patches in the same scene, but it can have a large performance benefit if so.

Parameters

<i>modelMatrix</i>	Receives an array of 16 doubles defining a 4x4 model matrix for the ocean patch. If set to NULL, an identity matrix is assumed.
--------------------	---

Returns

True if the matrix was set successfully.

22.8.3.43 `virtual bool TRITONAPI Triton::Ocean::SetPatchShader ( double time, int vertexStride, int positionOffset, bool doublePrecisionVertices, const double * modelMatrix = 0 )`
[virtual]

Sets the shaders and state necessary for rendering a user-provided patch of geometry as water.

Use this if you don't want Triton to draw an infinite ocean using [Ocean::Draw\(\)](#), but only want to draw your own water patches. All that's required is the rendering of vertex data containing 3D position data following this call, and a call to [Ocean::UnsetPatchShader\(\)](#) must be called after you've drawn your water geometry.

The geometry drawn should be flat, relative to sea level. Our shaders will displace the vertices rendered and apply all the textures and reflections to make it look like water. Remember to call [Ocean::SetDepth\(\)](#) if you are drawing an area of shallow water and don't want this patch to look like open ocean.

For best results, ensure your depth tests are set to "less than or equal" to ensure proper sorting of Triton's waves against each other.

Note that particle-based spray effects won't be rendered when you're drawing your own geometry, all this does is shade the geometry you draw following this call to make your geometry look like water. You'll still get foam, reflections, and refractions. This method only works with TESSENDORF waves.

If you are using a WGS84 or SPHERICAL CoordinateSystem, our shaders assume that your mesh is positioned relative to the center of the Earth being at 0, 0, 0. To preserve precision, your vertices when transformed by the modelMatrix provided should be relative to the camera position. Our shaders are compiled at runtime, so you may modify them if your data is of an unusual format. Look for the shaders ending with -patch.fx or -patch.glsl inside the Resources directory; these are used only by this method.

OpenGL users should ensure that your vertex array is bound prior to calling this method; DirectX11 users should set their vertex buffer with the input assembly stage prior to calling this method.

See also

[UnsetPatchShader\(\)](#)
[SetPatchMatrix\(\)](#)

Parameters

<i>time</i>	The simulated point in time to render, in seconds. Note that this is an absolute time which can be relative to any arbitrary point in time. It's not the delta time between frames.
-------------	---

<i>vertexStride</i>	The number of bytes between vertices in your vertex array. This is only used in OpenGL 3.2+ and DirectX11 to allow binding of your vertex array with the appropriate vertex attribute in our shaders.
<i>positionOffset</i>	The number of bytes from the start of each vertex to the x,y,z floating point position data. Used only in OpenGL 3.2+.
<i>doublePrecisionVertices</i>	Pass in true if your vertex data is "double" precision, or false for "float" single precision data. Only relevant in OpenGL 3.2 or newer.
<i>modelMatrix</i>	Receives an array of 16 doubles defining a 4x4 model matrix for the ocean patch. If set to NULL, an identity matrix is assumed.

Returns

True if the state was successfully set and drawing may continue.

22.8.3.44 void TRITONAPI Triton::Ocean::SetPlanarReflectionBlend (float *blendPercent*)

Sets the prominence of planar reflections on this ocean surface, if one was set using [Triton::Environment::SetPlanarReflectionMap\(\)](#).

Parameters

<i>percent</i>	The percent by which planar reflections will be blended into the reflected color of the water (0 - 1.0).
----------------	--

22.8.3.45 void TRITONAPI Triton::Ocean::SetQuality (OceanQuality *quality*)

Set a quality setting (GOOD, BETTER, or BEST.) Higher quality will result in finer wave resolution, but at lower performance.

Default value is GOOD. Changing the quality setting requires deleting and re-initializing most of Triton's internal objects, and so there will be a pause while this change is processed.

22.8.3.46 void TRITONAPI Triton::Ocean::SetRefractionColor (const Vector3 & *refractionColor*)

Modifies the color used for refracted light rays that go into deep water.

You can use this to modify the color of the water in areas that are not purely reflective.

Parameters

<i>refraction-Color</i>	the RGB color value of the deep water color; each component should be in the range 0-1.
-------------------------	---

22.8.3.47 `bool TRITONAPI Triton::Ocean::SprayEnabled ( ) const [inline]`

Returns if spray particle effects on breaking waves are enabled, which they are by default.

See also

[EnableSpray\(\)](#).

22.8.3.48 `void TRITONAPI Triton::Ocean::Unlock ( )`

Explicitly locks the ocean's mutex previously locked by [Ocean::Lock\(\)](#).

22.8.3.49 `virtual void TRITONAPI Triton::Ocean::UnsetPatchShader ( double time = 0.0f, const TBoundingBox * patchBounds = 0 ) [virtual]`

Restores the graphics state prior to a previous call to [Ocean::SetPatchShader\(\)](#).

Every call to [Ocean::SetPatchShader\(\)](#) must be matched with a call to [Ocean::UnsetPatchShader\(\)](#) following the drawing of any user-defined patches of water. If you want particle-based spray effects, you may optionally pass in a timestamp and bounding box for your patch to enable these effects.

Parameters

<i>time</i>	The current time, in seconds, used for particle animations. Optional.
<i>patch-Bounds</i>	A bounding box defining the bounds of your patch, used for particle animations. Optional.

See also

[SetPatchShader\(\)](#)

22.8.3.50 `virtual void TRITONAPI Triton::Ocean::UpdateSimulation ( double time ) [virtual]`

Updates the underlying wave simulation; calling this is optional and only necessary if you wish to perform physics updates from a pass or thread different from the rendering pass or thread.

If [UpdateSimulation\(\)](#) is not called prior to [Draw\(\)](#) or [SetPatchShader\(\)](#), then [Draw\(\)](#) or [SetPatchShader\(\)](#) will call it automatically. A mutex is enforced between this method and the [Draw\(\)](#), [SetPatchShader\(\)](#), and [GetHeight\(\)](#) methods.

Parameters

<i>time</i>	The simulated point in time to render, in seconds. Note that this is an absolute time which can be relative to any arbitrary point in time. It's not the delta time between frames.
-------------	---

The documentation for this class was generated from the following file:

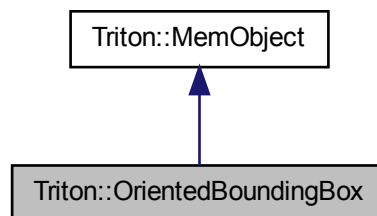
- C:/triton/trunk/Public Headers/[Ocean.h](#)

22.9 Triton::OrientedBoundingBox Class Reference

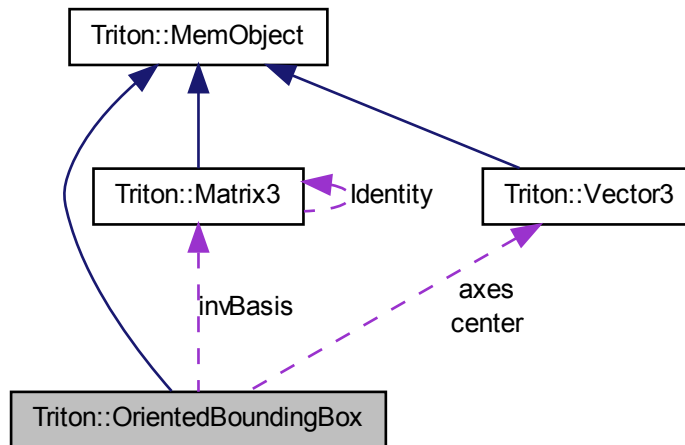
An oriented bounding box defined by a center point and three axes.

```
#include <OrientedBoundingBox.h>
```

Inheritance diagram for Triton::OrientedBoundingBox:



Collaboration diagram for Triton::OrientedBoundingBox:



Public Member Functions

- [OrientedBoundingBox](#) ()

Constructor.

- void [Set](#) (const [Vector3](#) &center, const [Vector3](#) &xExtent, const [Vector3](#) &yExtent, const [Vector3](#) &zExtent)

Define the OBB by a center point and vectors from center to extents in X, Y, and Z.

- bool [PointInBox](#) (const [Vector3](#) &point, double slop) const

Test if a point is enclosed by the box.

- void [RecomputeBasis](#) ()

Recomputes the basis used for PointInBox.

22.9.1 Detailed Description

An oriented bounding box defined by a center point and three axes.

22.9.2 Constructor & Destructor Documentation

22.9.2.1 Triton::OrientedBoundingBox::OrientedBoundingBox ()

Constructor.

22.9.3 Member Function Documentation

22.9.3.1 bool Triton::OrientedBoundingBox::PointInBox (const Vector3 & *point*, double *slop*) const

Test if a point is enclosed by the box.

22.9.3.2 void Triton::OrientedBoundingBox::Set (const Vector3 & *center*, const Vector3 & *xExtent*, const Vector3 & *yExtent*, const Vector3 & *zExtent*)

Define the OBB by a center point and vectors from center to extents in X, Y, and Z.

The documentation for this class was generated from the following file:

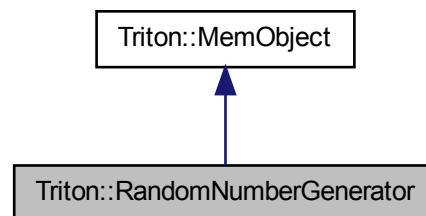
- C:/triton/trunk/Public Headers/[OrientedBoundingBox.h](#)

22.10 Triton::RandomNumberGenerator Class Reference

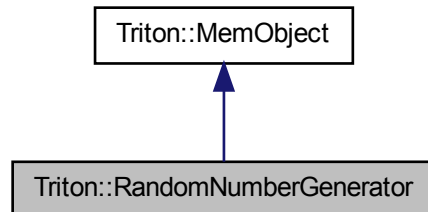
An interface for generating random numbers in Triton.

```
#include <RandomNumberGenerator.h>
```

Inheritance diagram for Triton::RandomNumberGenerator:



Collaboration diagram for Triton::RandomNumberGenerator:



Public Member Functions

- virtual double TRITONAPI [GetRandomDouble](#) (double start, double end) const =0
Return an evenly distributed random double-precision number within a given range.
- virtual int TRITONAPI [GetRandomInt](#) (int start, int end) const =0
Return an evenly distributed random integer within a given range.
- virtual void TRITONAPI [SetRandomSeed](#) (unsigned int seed)=0
Seeds the random number generator with a given value, to ensure consistent results.

22.10.1 Detailed Description

An interface for generating random numbers in Triton. Subclass this interface and pass an instance to [Environment::SetRandomNumberGenerator](#) in order to override Triton's default usage of `rand()`. This may be useful for enforcing deterministic behavior across several channels.

22.10.2 Member Function Documentation

22.10.2.1 virtual double TRITONAPI Triton::RandomNumberGenerator::GetRandomDouble (double start, double end) const [pure virtual]

Return an evenly distributed random double-precision number within a given range.

Parameters

<i>start</i>	The lowest value in the range
<i>end</i>	The highest value in the range

Returns

An evenly distributed random number within the range.

22.10.2.2 `virtual int TRITONAPI Triton::RandomNumberGenerator::GetRandomInt ( int start, int end ) const` [pure virtual]

Return an evenly distributed random integer within a given range.

Parameters

<i>start</i>	The lowest value in the range
<i>end</i>	The highest value in the range

Returns

An evenly distributed random number within the range.

22.10.2.3 `virtual void TRITONAPI Triton::RandomNumberGenerator::SetRandomSeed ( unsigned int seed )` [pure virtual]

Seeds the random number generator with a given value, to ensure consistent results.

Parameters

<i>seed</i>	A value used to seed the random number generator's sequence of psuedo-random numbers.
-------------	---

The documentation for this class was generated from the following file:

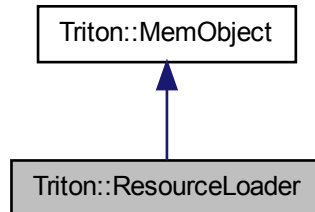
- C:/triton/trunk/Public Headers/[RandomNumberGenerator.h](#)

22.11 Triton::ResourceLoader Class Reference

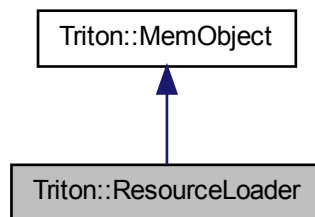
This class is used whenever Triton needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish.

```
#include <ResourceLoader.h>
```

Inheritance diagram for Triton::ResourceLoader:



Collaboration diagram for Triton::ResourceLoader:



Public Member Functions

- [ResourceLoader](#) (const char *resourceDirPath, bool useAddDllDirectory=false)

Constructor.

- virtual [~ResourceLoader](#) ()

Virtual destructor; frees the memory of the resource path string.

- virtual void TRITONAPI [SetResourceDirPath](#) (const char *path, bool useAddDllDirectory=false)

Sets the path to the Triton resources folder, which will be pre-pended to all resource filenames passed into [LoadResource\(\)](#).

- virtual const char *TRITONAPI [GetResourceDirPath](#) () const
Retrieves the path set by [SetResourceDirPath](#)().
- virtual bool TRITONAPI [LoadResource](#) (const char *pathName, char *&data, unsigned int &dataLen, bool text)
Load a resource from mass storage; the default implementation uses the POSIX functions [fopen](#)(), [fread](#)(), and [fclose](#)() to do this, but you may override this method to load resources however you wish.
- virtual void TRITONAPI [FreeResource](#) (char *data)
Frees the resource data memory that was returned from [LoadResource](#)().

22.11.1 Detailed Description

This class is used whenever Triton needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish. If you have your own system of packed files, you can include Triton's resources directory into it and implement your own [ResourceLoader](#) to access our resources within your pack files.

22.11.2 Constructor & Destructor Documentation

22.11.2.1 **Triton::ResourceLoader::ResourceLoader** (const char * *resourceDirPath*, bool *useAddDllDirectory* = false)

Constructor.

Parameters

<i>resourceDirPath</i>	The path to Triton's resources folder. Avoid using relative paths if at all possible.
<i>useAddDllDirectory</i>	Only applicable to Windows; controls whether we attempt to add the resources/dll directory to the DLL search path using the Windows AddDllDirectory function instead of SetDLLDirectory . AddDllDirectory is less destructive to the existing DLL search path, but is not well supported on older systems.

22.11.3 Member Function Documentation

22.11.3.1 **virtual void TRITONAPI Triton::ResourceLoader::FreeResource** (char * *data*)
[virtual]

Frees the resource data memory that was returned from [LoadResource](#)().

The data pointer will be invalid following this call.

22.11.3.2 `virtual bool TRITONAPI Triton::ResourceLoader::LoadResource ( const char * pathName, char *& data, unsigned int & dataLen, bool text ) [virtual]`

Load a resource from mass storage; the default implementation uses the POSIX functions `fopen()`, `fread()`, and `fclose()` to do this, but you may override this method to load resources however you wish.

The caller is responsible for calling [FreeResource\(\)](#) when it's done consuming the resource data in order to free its memory.

Parameters

<i>pathName</i>	The path to the desired resource, relative to the location of the resources folder previously specified in SetResourceDirPath() .
<i>data</i>	A reference to a <code>char *</code> that will return the resource's data upon a successful load.
<i>dataLen</i>	A reference to an unsigned int that will return the number of bytes loaded upon a successful load.
<i>text</i>	True if the resource is a text file, such as a shader. If true, a terminating null character will be appended to the resulting data and the file will be opened in text mode.

Returns

True if the resource was located and loaded successfully, false otherwise.

See also

[SetResourceDirPath](#)

22.11.3.3 `virtual void TRITONAPI Triton::ResourceLoader::SetResourceDirPath ( const char * path, bool useAddDllDirectory = false ) [virtual]`

Sets the path to the Triton resources folder, which will be pre-pended to all resource filenames passed into [LoadResource\(\)](#).

This method also calls the Win32 function `AddDllDirectory` in order to add the dll subdirectory to the application's DLL search path. This should be a fully qualified path and not a relative one if at all possible.

Parameters

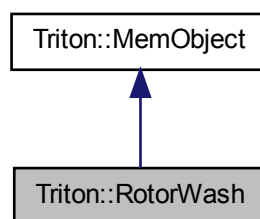
<i>path</i>	The path to Triton's Resources folder; avoid using relative paths.
<i>useAddDllDirectory</i>	On Windows, controls whether we attempt to use the <code>AddDllDirectory</code> function instead of <code>SetDllDirectory</code> in order to add our runtime dependencies into the DLL search path.

The documentation for this class was generated from the following file:

- `C:/triton/trunk/Public Headers/ResourceLoader.h`

A **RotorWash** object will generate spray and circular waves on the ocean surface in the direction it is pointing.

Inheritance diagram for Triton::RotorWash:



```

graph LR
    Triton_Environment[Triton::Environment]
    Triton_ResourceLoader[Triton::ResourceLoader]
    Triton_Matrix3[Triton::Matrix3]
    Triton_Matrix4[Triton::Matrix4]
    Triton_BreakingWavesParameters[Triton::BreakingWavesParameters]
    Triton_Matrix3_2[Triton::Matrix3]
    Triton_Vector3[Triton::Vector3]
    Triton_RandomNumberGenerator[Triton::RandomNumberGenerator]
    Triton_MemObject[Triton::MemObject]
    Triton_Ocean[Triton::Ocean]
    Triton_RigidBody[Triton::RigidBody]

    Triton_Environment -- resource_loader --> Triton_ResourceLoader
    Triton_Environment -- planeReflectionMapMatrix --> Triton_Matrix3
    Triton_Environment -- envMapMatrix --> Triton_Matrix3
    Triton_Environment -- breakingWavesParameters --> Triton_BreakingWavesParameters
    Triton_Environment -- lightColor --> Triton_Matrix3_2
    Triton_Environment -- belowWaterFogColor --> Triton_Matrix3_2
    Triton_Environment -- ambientColor --> Triton_Matrix3_2
    Triton_Environment -- lightDirection --> Triton_Matrix3_2
    Triton_Environment -- aboveWaterFogColor --> Triton_Matrix3_2
    Triton_Environment -- breakingWaveMapMatrix --> Triton_Matrix4
    Triton_Environment -- heightMapMatrix --> Triton_Matrix4
    Triton_Environment -- defaultRandomNumberGenerator --> Triton_RandomNumberGenerator
    Triton_Environment -- randomNumberGenerator --> Triton_RandomNumberGenerator
    Triton_Environment -- lastExitSourcePosition --> Triton_MemObject
    Triton_Environment -- position --> Triton_MemObject
    Triton_Environment -- lastExitPosition --> Triton_MemObject
    Triton_Environment -- lastEnvPosition --> Triton_MemObject
    Triton_Environment -- lastPosition --> Triton_MemObject

    Triton_ResourceLoader --> Triton_Matrix3
    Triton_Matrix3 --> Triton_Matrix4
    Triton_BreakingWavesParameters --> Triton_Matrix4
    Triton_Matrix4 --> Triton_Matrix3
    Triton_Ocean --> Triton_RigidBody
    Triton_RigidBody --> Triton_Vector3
    Triton_Vector3 --> Triton_RandomNumberGenerator
    Triton_RandomNumberGenerator --> Triton_MemObject
    Triton_MemObject --> Triton_Environment
  
```

- **RotorWash** (**Ocean** *pOcean, double pRotorDiameter, bool pSprayEffects=false, bool pUseDecals=false)

Generated on Tue Jan 27 2015 10:10:01 for Triton by Doxygen

- void TRITONAPI [Update](#) (const [Vector3](#) &pPosition, const [Vector3](#) &pDirection, double pVelocity, double pTime)

For any active [RotorWash](#), this should be called every frame to update its position and velocity.

- [Vector3](#) TRITONAPI [GetPosition](#) () const
Retrieves the world position of the [RotorWash](#).

- double TRITONAPI [GetVelocity](#) () const
Retrieves the airspeed velocity of the [RotorWash](#).

22.12.1 Detailed Description

A [RotorWash](#) object will generate spray and circular waves on the ocean surface in the direction it is pointing.

22.12.2 Constructor & Destructor Documentation

22.12.2.1 [Triton::RotorWash::RotorWash](#) ([Ocean](#) * pOcean, double pRotorDiameter, bool pSprayEffects = false, bool pUseDecals = false)

Construct a [RotorWash](#) with the same [Triton::Ocean](#) it will be associated with.

Parameters

<i>pOcean</i>	The Triton::Ocean object you will associate this RotorWash with. A common error is to create a RotorWash before the Ocean has been created, so make sure this is a valid, non-null pointer.
<i>pRotorDiameter</i>	The diameter of the rotor blades in world units.
<i>pSprayEffects</i>	Whether you wish this wash to emit spray particles.
<i>pUseDecals</i>	Whether decal textures should be used to provide a more detailed, but costly effect.

22.12.3 Member Function Documentation

22.12.3.1 [Vector3](#) TRITONAPI [Triton::RotorWash::GetPosition](#) () const [inline]

Retrieves the world position of the [RotorWash](#).

Returns

The world position of the [RotorWash](#), as last specified by [Triton::RotorWash::Update\(\)](#).

22.12.3.2 `double TRITONAPI Triton::RotorWash::GetVelocity ( ) const` `[inline]`

Retrieves the airspeed velocity of the [RotorWash](#).

Returns

The air velocity of the [RotorWash](#) in world units per second, as last specified by [Triton::RotorWash::Update\(\)](#).

22.12.3.3 `void TRITONAPI Triton::RotorWash::Update ( const Vector3 & pPosition, const Vector3 & pDirection, double pVelocity, double pTime )`

For any active [RotorWash](#), this should be called every frame to update its position and velocity.

No wash will be generated until this is called.

Parameters

<i>pPosition</i>	The position of the rotor, in world coordinates.
<i>pDirection</i>	A normalized direction vector indicating the direction the rotors are pointing down toward.
<i>pVelocity</i>	The velocity of the wind generating the wake, in world units per second.
<i>pTime</i>	The current simulated time sample, in seconds. This may be relative to any reference point in time, as long as that reference point is consistent among the multiple calls to Update() .

The documentation for this class was generated from the following file:

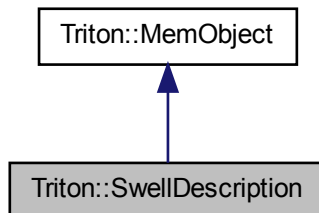
- C:/triton/trunk/Public Headers/[RotorWash.h](#)

22.13 Triton::SwellDescription Class Reference

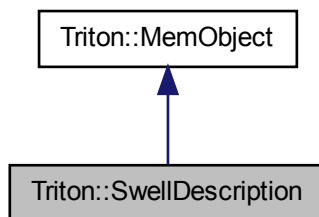
A structure containing a description of a swell in addition to local wind waves (from a distant storm perhaps.)

```
#include <Environment.h>
```

Inheritance diagram for Triton::SwellDescription:



Collaboration diagram for Triton::SwellDescription:



Public Attributes

- float [height](#)
Wavelength in world units, from peak to peak.
- float [direction](#)
Wave height, from peak to trough.
- float [phase](#)
Direction in radians.

22.13.1 Detailed Description

A structure containing a description of a swell in addition to local wind waves (from a distant storm perhaps.)

The documentation for this class was generated from the following file:

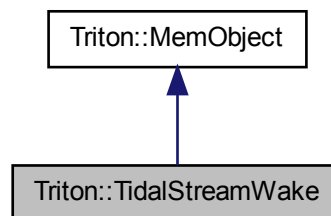
- C:/triton/trunk/Public Headers/[Environment.h](#)

22.14 Triton::TidalStreamWake Class Reference

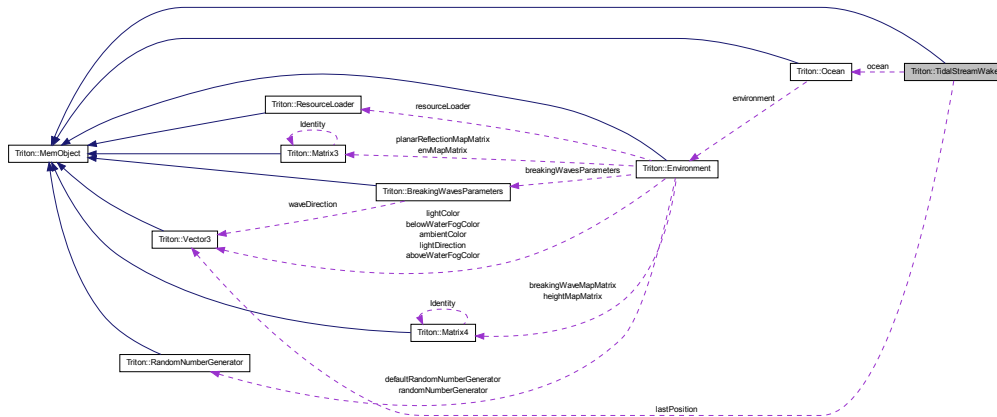
An static wake pointing in a given direction at a fixed location, for example from a buoy or bridge pile in a current.

```
#include <TidalStreamWake.h>
```

Inheritance diagram for Triton::TidalStreamWake:



Collaboration diagram for Triton::TidalStreamWake:



Public Member Functions

- **TidalStreamWake** (**Ocean** *pOcean, double pSize, double pDraft, double pWaveMax=1.0, double pOffset=0, bool pUseDecals=true, bool pUseDisplacement=true)

Construct a `TidalStreamWake` with the same `Triton::Ocean` it will be associated with.

- void TRITONAPI Update (const **Vector3** &pPosition, const **Vector3** &pDirection, double pVelocity, double pTime)

For any active *TidalStreamWake*, this should be called every frame to update its position and velocity.

22.14.1 Detailed Description

An static wake pointing in a given direction at a fixed location, for example from a buoy or bridge pile in a current.

22.14.2 Constructor & Destructor Documentation

```
22.14.2.1 Triton::TidalStreamWake::TidalStreamWake ( Ocean * pOcean, double pSize,
double pDraft, double pWaveMax = 1.0, double pOffset = 0, bool pUseDecals =
true, bool pUseDisplacement = true )
```

Construct a `TidalStreamWake` with the same `Triton::Ocean` it will be associated with.

Parameters

<i>pOcean</i>	The Triton::Ocean object you will associate this TidalStreamWake with. A common error is to create a TidalStreamWake before the Ocean is created, so make sure this is a valid, non-null pointer.
<i>pSize</i>	The length of the wake generated.
<i>pDraft</i>	The distance underwater this object extends.
<i>pWaveMax</i>	The maximum wake wave amplitude generated by this object.
<i>pOffset</i>	An offset from the object's position to the front of the wake wave.
<i>pUseDecals</i>	Whether a decal texture should be applied over the wake for higher resolution appearance from above.
<i>pUseDisplacement</i>	Whether the wake will displace the ocean surface in 3D or not. For very small tidal stream wakes, anomalies may result due to the limited resolution of the underlying water mesh.

22.14.3 Member Function Documentation

22.14.3.1 void TRITONAPI Triton::TidalStreamWake::Update (const Vector3 & *pPosition*, const Vector3 & *pDirection*, double *pVelocity*, double *pTime*)

For any active [TidalStreamWake](#), this should be called every frame to update its position and velocity.

No wake will be generated until this is called.

Parameters

<i>pPosition</i>	The position of the object producing the wake, in world coordinates.
<i>pDirection</i>	A normalized direction vector indicating the direction of the current or tidal stream.
<i>pVelocity</i>	The velocity of the current generating the wake, in meters per second. This will influence the height of the wake, up to the maximum amplitude specified in the constructor.
<i>pTime</i>	The current simulated time sample, in seconds. This may be relative to any reference point in time, as long as that reference point is consistent among the multiple calls to Update() .

The documentation for this class was generated from the following file:

- C:/triton/trunk/Public Headers/[TidalStreamWake.h](#)

22.15 Triton::Utils Class Reference

A collection of static utility methods used by Triton.

```
#include <TritonCommon.h>
```

Static Public Member Functions

- static void TRITONAPI [SetDebugMsgCB](#) (void(*debug_msg_cb)(const char *))

Allows the client to plug in a routine which will be called to print/log [DebugMsg\(\)](#) calls when "enable-debug-messages" is set, instead of using the internal default logging behavior.

- static void TRITONAPI [DebugMsg](#) (const char *debugMessage)

Prints out an error message to the debugger's output window only if the setting enable-debug-messages is set to 'yes' in resources/Triton.config.

- static void TRITONAPI [ClearGLErrors](#) ()

Clears any existing OpenGL error codes so we may test for new ones.

- static bool TRITONAPI [PrintGLErrors](#) (const char *file, int line)

Prints out (and clears) any existing GL errors using [Utils::DebugMsg](#).

- static TRITON_STRING TRITONAPI [GetDLLPath](#) ()

Returns the resources subdirectory where our FFT implementation DLL's live for this specific build's compiler version and platform.

- static TRITON_STRING TRITONAPI [GetDLLSuffix](#) ()

Retrieves the suffix on the FFT implementation DLL's for this specific build flavor's runtime library.

- static TRITON_STRING TRITONAPI [GetDLLExtension](#) ()

Retrieves the DLL extension.

- static struct _D3DXMACRO *TRITONAPI [GetDX9Macros](#) (IDirect3DDevice9 *device, bool hdr, bool breakingWaves)

Returns HLSL preprocessor defines for DirectX9, indicating the shader models available to our shaders.

- static TRITON_STRING TRITONAPI [GetWaterShaderFileName](#) (const [Environment](#) *env, bool tessendorf, bool fragment, bool patch)

Returns the filename for the main water shaders, which may be overridden via config.

- static TRITON_STRING TRITONAPI [GetParticleShaderFileName](#) (const [Environment](#) *env, bool fragment)

Returns the filename for the particle shaders, which may be overridden via config.

- static TRITON_STRING TRITONAPI [GetUserShaderFileName](#) ()

Returns the filename for the user-defined fragment shader functions (OpenGL only; see the Resources/user-functions.glsl file.)

- static TRITON_STRING TRITONAPI [GetUserVertShaderFileName](#) ()

Returns the filename for the user-defined vertex shader functions (OpenGL only; see the Resources/user-vert-functions.glsl file.)

- static TRITON_STRING TRITONAPI [GetDecalShaderFileName](#) (const [Environment](#) *env, bool fragment)

Returns the filename for the volumetric deferred decal shaders, which may be overridden via config.

- static TRITON_STRING TRITONAPI [GetGodRayShaderFileName](#) (const [Environment](#) *env, bool fragment)

Returns the filename for the god ray shaders, which may be overridden via config.

22.15.1 Detailed Description

A collection of static utility methods used by Triton.

22.15.2 Member Function Documentation

22.15.2.1 static void TRITONAPI Triton::Utils::ClearGLErrors () [static]

Clears any existing OpenGL error codes so we may test for new ones.

22.15.2.2 static TRITON_STRING TRITONAPI Triton::Utils::GetDecalShaderFileName (const Environment * env, bool fragment) [static]

Returns the filename for the volumetric deferred decal shaders, which may be overridden via config.

22.15.2.3 static TRITON_STRING TRITONAPI Triton::Utils::GetDLLExtension () [inline, static]

Retrieves the DLL extension.

22.15.2.4 static TRITON_STRING TRITONAPI Triton::Utils::GetDLLPath () [inline, static]

Returns the resources subdirectory where our FFT implementation DLL's live for this specific build's compiler version and platform.

22.15.2.5 static TRITON_STRING TRITONAPI Triton::Utils::GetDLLSuffix () [static]

Retrieves the suffix on the FFT implementation DLL's for this specific build flavor's runtime library.

22.15.2.6 `static struct _D3DXMACRO* TRITONAPI Triton::Utils::GetDX9Macros ( IDirect3DDevice9 * device, bool hdr, bool breakingWaves ) [static, read]`

Returns HLSL preprocessor defines for DirectX9, indicating the shader models available to our shaders.

Returns

NULL if minimum system requirements for DX9 are not present, a null-terminated D3DXMACRO array otherwise.

22.15.2.7 `static TRITON_STRING TRITONAPI Triton::Utils::GetGodRayShaderFileName ( const Environment * env, bool fragment ) [static]`

Returns the filename for the god ray shaders, which may be overridden via config.

22.15.2.8 `static TRITON_STRING TRITONAPI Triton::Utils::GetParticleShaderFileName ( const Environment * env, bool fragment ) [static]`

Returns the filename for the particle shaders, which may be overridden via config.

22.15.2.9 `static TRITON_STRING TRITONAPI Triton::Utils::GetWaterShaderFileName ( const Environment * env, bool tessendorf, bool fragment, bool patch ) [static]`

Returns the filename for the main water shaders, which may be overridden via config.

22.15.2.10 `static bool TRITONAPI Triton::Utils::PrintGLErrors ( const char * file, int line ) [static]`

Prints out (and clears) any existing GL errors using [Utils::DebugMsg](#).

Returns

True if no error code was present.

22.15.2.11 `static void TRITONAPI Triton::Utils::SetDebugMsgCB ( void (*)(const char *) debug_msg_cb ) [static]`

Allows the client to plug in a routine which will be called to print/log [DebugMsg\(\)](#) calls when "enable-debug-messages" is set, instead of using the internal default logging behavior.

The documentation for this class was generated from the following file:

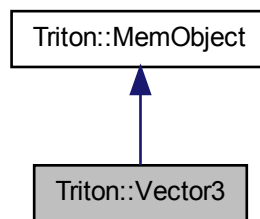
- C:/triton/trunk/Public Headers/[TritonCommon.h](#)

22.16 Triton::Vector3 Class Reference

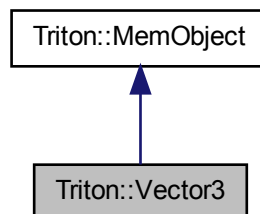
A 3D double-precision Vector class and its operations.

```
#include <Vector3.h>
```

Inheritance diagram for Triton::Vector3:



Collaboration diagram for Triton::Vector3:



Public Member Functions

- [Vector3](#) ()
Default constructor, initializes to (0,0,0).
- [Vector3](#) (double px, double py, double pz)
Constructs a [Vector3](#) with the given x,y,z coordinates.

- [Vector3](#) (const double *p)
Constructs a Vector 3 from a pointer to 3 doubles.
- [Vector3](#) (const [Vector3](#) &v)
Copy constructor.
- double TRITONAPI [Length](#) () const
Returns the length of the vector.
- double TRITONAPI [SquaredLength](#) () const
Returns the squared length of the vector, which is faster than computing the length.
- void TRITONAPI [Normalize](#) ()
Scales the vector to be of length 1.0.
- double TRITONAPI [Dot](#) (const [Vector3](#) &v) const
Determines the dot product between this vector and another, and returns the result.
- [Vector3](#) TRITONAPI [Cross](#) (const [Vector3](#) &v) const
Determines the cross product between this vector and another, and returns the result.
- [Vector3](#) TRITONAPI [operator*](#) (double n) const
Scales each x,y,z value of the vector by a constant n, and returns the result.
- [Vector3](#) TRITONAPI [operator*](#) (const [Vector3](#) &v) const
Multiplies the components of two vectors together, and returns the result.
- [Vector3](#) TRITONAPI [operator+](#) (double n) const
Adds a constant n to each component of the vector, and returns the result.
- [Vector3](#) TRITONAPI [operator-](#) (const [Vector3](#) &v) const
Subtracts the specified vector from this vector, and returns the result.
- [Vector3](#) TRITONAPI [operator+](#) (const [Vector3](#) &v) const
Adds this vector to the specified vector, and returns the result.
- bool TRITONAPI [operator==](#) (const [Vector3](#) &v) const
Tests if two vectors are exactly equal.
- bool TRITONAPI [operator!=](#) (const [Vector3](#) &v) const
Test if two vectors are not exactly equal.
- void TRITONAPI [Serialize](#) (std::ostream &s) const
Write this vector's data to a file.
- void TRITONAPI [Unserialize](#) (std::istream &s)
Restore this vector from a file.

Public Attributes

- double [x](#)

Data members x,y,z public for convenience.

22.16.1 Detailed Description

A 3D double-precision Vector class and its operations.

22.16.2 Constructor & Destructor Documentation

22.16.2.1 `Triton::Vector3::Vector3 ( )` `[inline]`

Default constructor, initializes to (0,0,0).

22.16.2.2 `Triton::Vector3::Vector3 ( double px, double py, double pz )` `[inline]`

Constructs a [Vector3](#) with the given x,y,z coordinates.

22.16.3 Member Function Documentation

22.16.3.1 `Vector3 TRITONAPI Triton::Vector3::Cross ( const Vector3 & v ) const` `[inline]`

Determines the cross product between this vector and another, and returns the result.

22.16.3.2 `double TRITONAPI Triton::Vector3::Dot ( const Vector3 & v ) const` `[inline]`

Determines the dot product between this vector and another, and returns the result.

22.16.3.3 `double TRITONAPI Triton::Vector3::Length ( ) const` `[inline]`

Returns the length of the vector.

22.16.3.4 `void TRITONAPI Triton::Vector3::Normalize ( )` `[inline]`

Scales the vector to be of length 1.0.

22.16.3.5 `bool TRITONAPI Triton::Vector3::operator!= ( const Vector3 & v ) const` `[inline]`

Test if two vectors are not exactly equal.

22.16.3.6 Vector3 TRITONAPI Triton::Vector3::operator* (double *n*) const [inline]

Scales each x,y,z value of the vector by a constant *n*, and returns the result.

22.16.3.7 Vector3 TRITONAPI Triton::Vector3::operator* (const Vector3 & *v*) const
[inline]

Multiplies the components of two vectors together, and returns the result.

22.16.3.8 Vector3 TRITONAPI Triton::Vector3::operator+ (double *n*) const [inline]

Adds a constant *n* to each component of the vector, and returns the result.

22.16.3.9 Vector3 TRITONAPI Triton::Vector3::operator+ (const Vector3 & *v*) const
[inline]

Adds this vector to the specified vector, and returns the result.

22.16.3.10 Vector3 TRITONAPI Triton::Vector3::operator- (const Vector3 & *v*) const
[inline]

Subtracts the specified vector from this vector, and returns the result.

22.16.3.11 bool TRITONAPI Triton::Vector3::operator== (const Vector3 & *v*) const
[inline]

Tests if two vectors are exactly equal.

22.16.3.12 void TRITONAPI Triton::Vector3::Serialize (std::ostream & *s*) const [inline]

Write this vector's data to a file.

22.16.3.13 double TRITONAPI Triton::Vector3::SquaredLength () const [inline]

Returns the squared length of the vector, which is faster than computing the length.

22.16.3.14 void TRITONAPI Triton::Vector3::Unserialize (std::istream & *s*) [inline]

Restore this vector from a file.

22.16.4 Member Data Documentation

22.16.4.1 double Triton::Vector3::x

Data members x,y,z public for convenience.

The documentation for this class was generated from the following file:

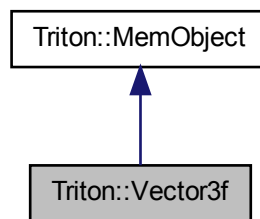
- C:/triton/trunk/Public Headers/[Vector3.h](#)

22.17 Triton::Vector3f Class Reference

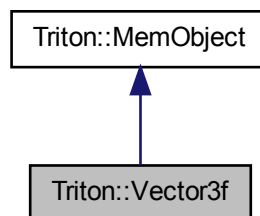
A 3D single-precision vector class, and its operations.

```
#include <Vector3.h>
```

Inheritance diagram for Triton::Vector3f:



Collaboration diagram for Triton::Vector3f:



Public Member Functions

- [Vector3f](#) ()
Default constructor; does not initialize the vector.
- [Vector3f](#) (const [Triton::Vector3](#) &*v*)
Construct a single precision vector from a double precision one.
- [Vector3f](#) (float *px*, float *py*, float *pz*)
Constructs a [Vector3f](#) from the specified single-precision floating point x, y, and z values.
- float [TRITONAPI Length](#) ()
Returns the length of this vector.
- void [TRITONAPI Normalize](#) ()
Scales the vector to be of length 1.0.
- double [TRITONAPI Dot](#) (const [Vector3f](#) &*v*) const
Returns the dot product of this vector with the specified [Vector3](#).
- [Vector3f](#) [TRITONAPI operator-](#) (const [Vector3f](#) &*v*) const
Subtracts the specified vector from this vector, and returns the result.
- [Vector3f](#) [TRITONAPI operator+](#) (const [Vector3f](#) &*v*) const
Adds this vector to the specified vector, and returns the result.
- [Vector3f](#) [TRITONAPI operator*](#) (const [Vector3f](#) &*v*) const
Multiplies the components of two vectors together, and returns the result.
- [Vector3f](#) [TRITONAPI operator*](#) (float *n*) const
Scales each x,y,z value of the vector by a constant n, and returns the result.

Public Attributes

- float [x](#)
Data members x, y, z are public for convenience.

22.17.1 Detailed Description

A 3D single-precision vector class, and its operations.

22.17.2 Constructor & Destructor Documentation

22.17.2.1 `Triton::Vector3f::Vector3f ( )` `[inline]`

Default constructor; does not initialize the vector.

22.17.2.2 `Triton::Vector3f::Vector3f ( const Triton::Vector3 & v )` `[inline]`

Construct a single precision vector from a double precision one.

22.17.2.3 `Triton::Vector3f::Vector3f ( float px, float py, float pz )` `[inline]`

Constructs a [Vector3f](#) from the specified single-precision floating point x, y, and z values.

22.17.3 Member Function Documentation

22.17.3.1 `double TRITONAPI Triton::Vector3f::Dot ( const Vector3f & v ) const` `[inline]`

Returns the dot product of this vector with the specified [Vector3](#).

22.17.3.2 `float TRITONAPI Triton::Vector3f::Length ( )` `[inline]`

Returns the length of this vector.

22.17.3.3 `void TRITONAPI Triton::Vector3f::Normalize ( )` `[inline]`

Scales the vector to be of length 1.0.

22.17.3.4 `Vector3f TRITONAPI Triton::Vector3f::operator* ( const Vector3f & v ) const` `[inline]`

Multiplies the components of two vectors together, and returns the result.

22.17.3.5 `Vector3f TRITONAPI Triton::Vector3f::operator* ( float n ) const` `[inline]`

Scales each x,y,z value of the vector by a constant n, and returns the result.

22.17.3.6 `Vector3f TRITONAPI Triton::Vector3f::operator+ ( const Vector3f & v ) const` `[inline]`

Adds this vector to the specified vector, and returns the result.

22.17.3.7 `Vector3f` TRITONAPI Triton::Vector3f::operator- (const Vector3f & v) const
[inline]

Subtracts the specified vector from this vector, and returns the result.

22.17.4 Member Data Documentation

22.17.4.1 `float` Triton::Vector3f::x

Data members x, y, z are public for convenience.

The documentation for this class was generated from the following file:

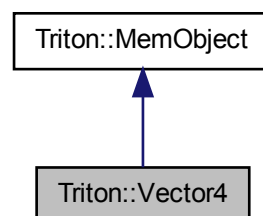
- C:/triton/trunk/Public Headers/[Vector3.h](#)

22.18 Triton::Vector4 Class Reference

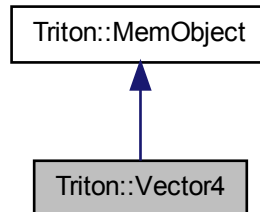
A simple double-precision 4D vector class with no operations defined.

```
#include <Vector4.h>
```

Inheritance diagram for Triton::Vector4:



Collaboration diagram for Triton::Vector4:



Public Member Functions

- **Vector4** (double px, double py, double pz, double pw)
*Constructs a **Vector4** from the given x, y, z, and w values.*
- **Vector4** (const **Vector3** &v3)
*Constructs a **Vector4** from a **Vector3**, setting w to 1.*
- **Vector4** ()
*Default constructor; initializes the **Vector4** to (0, 0, 0, 1)*
- double TRITONAPI **Dot** (const **Vector4** &v) const
Determines the dot product between this vector and another, and returns the result.
- **Vector4** TRITONAPI **operator*** (double n) const
Scales each x,y,z value of the vector by a constant n, and returns the result.
- **Vector4** TRITONAPI **operator*** (const **Vector4** &v) const
Multiplies the components of two vectors together, and returns the result.
- **Vector4** TRITONAPI **operator+** (double n) const
Adds a constant n to each component of the vector, and returns the result.
- **Vector4** TRITONAPI **operator-** (const **Vector4** &v) const
Subtracts the specified vector from this vector, and returns the result.
- **Vector4** TRITONAPI **operator+** (const **Vector4** &v) const
Adds this vector to the specified vector, and returns the result.

Public Attributes

- double [x](#)

The x, y, z, and w data members are public for convenience.

22.18.1 Detailed Description

A simple double-precision 4D vector class with no operations defined. Essentially a struct with constructors.

22.18.2 Constructor & Destructor Documentation

22.18.2.1 `Triton::Vector4::Vector4 ( double px, double py, double pz, double pw )`
`[inline]`

Constructs a [Vector4](#) from the given x, y, z, and w values.

22.18.2.2 `Triton::Vector4::Vector4 ( const Vector3 & v3 )` `[inline]`

Constructs a [Vector4](#) from a [Vector3](#), setting w to 1.

22.18.3 Member Function Documentation

22.18.3.1 `double TRITONAPI Triton::Vector4::Dot ( const Vector4 & v ) const` `[inline]`

Determines the dot product between this vector and another, and returns the result.

22.18.3.2 `Vector4 TRITONAPI Triton::Vector4::operator* ( double n ) const` `[inline]`

Scales each x,y,z value of the vector by a constant n, and returns the result.

22.18.3.3 `Vector4 TRITONAPI Triton::Vector4::operator* ( const Vector4 & v ) const`
`[inline]`

Multiplies the components of two vectors together, and returns the result.

22.18.3.4 `Vector4 TRITONAPI Triton::Vector4::operator+ ( double n ) const` `[inline]`

Adds a constant n to each component of the vector, and returns the result.

22.18.3.5 `Vector4 TRITONAPI Triton::Vector4::operator+ ( const Vector4 & v ) const`
[inline]

Adds this vector to the specified vector, and returns the result.

22.18.3.6 `Vector4 TRITONAPI Triton::Vector4::operator- ( const Vector4 & v ) const`
[inline]

Subtracts the specified vector from this vector, and returns the result.

22.18.4 Member Data Documentation

22.18.4.1 `double Triton::Vector4::x`

The x, y, z, and w data members are public for convenience.

The documentation for this class was generated from the following file:

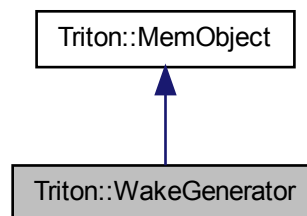
- C:/triton/trunk/Public Headers/[Vector4.h](#)

22.19 Triton::WakeGenerator Class Reference

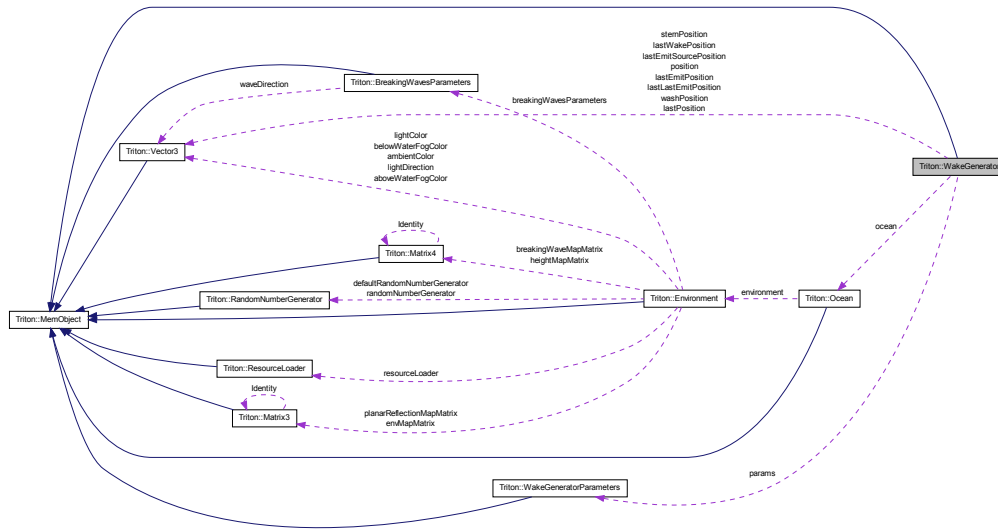
A [WakeGenerator](#) represents an object on the water that generates a wake as it moves, such as a ship.

```
#include <WakeGenerator.h>
```

Inheritance diagram for Triton::WakeGenerator:



Collaboration diagram for Triton::WakeGenerator:



Public Member Functions

- **WakeGenerator** (**Ocean** *pOcean, const **WakeGeneratorParameters** &parameters)

Construct a **WakeGenerator** with the same **Triton::Ocean** it will be associated with.

- void TRITONAPI **Update** (const **Vector3** &pPosition, const **Vector3** &pDirection, double pVelocity, double pTime)

For any active **WakeGenerator**, this should be called every frame to update its position and velocity.

- void **ClearWakes** ()

Clears all previously emitted wakes from the **Ocean**.

- **Vector3** TRITONAPI **GetPosition** () const

Retrieves the world position of the **WakeGenerator**.

- **Vector3** TRITONAPI **GetSternPosition** () const

Retrieves the world position of the point where stern wakes originate from.

- double TRITONAPI **GetVelocity** () const

Retrieves the velocity of the **WakeGenerator**.

- bool TRITONAPI **HasPropWash** () const

Retrieves whether propeller backwash was enabled for this wake generator.

- void TRITONAPI [SetLODDistance](#) (double distance)
Sets the distance at which the number of visible wake waves will be halved.
- double TRITONAPI [GetLODDistance](#) () const
Retrieves the LOD distance set in [Triton::WakeGenerator::SetLODDistance\(\)](#), or 0 if LOD is disabled.
- const [WakeGeneratorParameters](#) &TRITONAPI [GetParameters](#) () const
Returns the current parameters for this [WakeGenerator](#).
- void TRITONAPI [SetParameters](#) (const [WakeGeneratorParameters](#) &parameters)
Set this [WakeGenerator](#)'s parameters using the [WakeGeneratorParameters](#) provided.

22.19.1 Detailed Description

A [WakeGenerator](#) represents an object on the water that generates a wake as it moves, such as a ship. Simply call [Triton::WakeGenerator::Update\(\)](#) to move the object and generate a realistic wake behind it. Any [WakeGenerator](#) moving at constant velocity will generate a wake of 19.46 degrees behind it, but acceleration, deceleration, and curved paths are all handled properly as well.

22.19.2 Constructor & Destructor Documentation

22.19.2.1 [Triton::WakeGenerator::WakeGenerator](#) ([Ocean](#) * *pOcean*, const [WakeGeneratorParameters](#) & *parameters*)

Construct a [WakeGenerator](#) with the same [Triton::Ocean](#) it will be associated with.

This form of the constructor takes a [WakeGeneratorParameters](#) class instead of a long list of parameters, and is the preferred constructor to ensure code readability and prevent errors from misplaced parameters.

Parameters

<i>pOcean</i>	The Triton::Ocean object you will associate this WakeGenerator with. A common error is to create a WakeGenerator before the Ocean has been created, so make sure this is a valid, non-null pointer.
<i>parameters</i>	A Triton::WakeGeneratorParameters object containing the properties of this WakeGenerator .

22.19.3 Member Function Documentation

22.19.3.1 void Triton::WakeGenerator::ClearWakes ()

Clears all previously emitted wakes from the [Ocean](#).

22.19.3.2 double TRITONAPI Triton::WakeGenerator::GetLODDistance () const [inline]

Retrieves the LOD distance set in [Triton::WakeGenerator::SetLODDistance\(\)](#), or 0 if LOD is disabled.

22.19.3.3 const WakeGeneratorParameters& TRITONAPI Triton::WakeGenerator::GetParameters () const

Returns the current parameters for this [WakeGenerator](#).

22.19.3.4 Vector3 TRITONAPI Triton::WakeGenerator::GetPosition () const [inline]

Retrieves the world position of the [WakeGenerator](#).

Returns

The world position of the [WakeGenerator](#), as last specified by [Triton::WakeGenerator::Update\(\)](#).

22.19.3.5 Vector3 TRITONAPI Triton::WakeGenerator::GetSternPosition () const [inline]

Retrieves the world position of the point where stern wakes originate from.

22.19.3.6 double TRITONAPI Triton::WakeGenerator::GetVelocity () const [inline]

Retrieves the velocity of the [WakeGenerator](#).

Returns

The velocity of the [WakeGenerator](#) in world units per second, as last specified by [Triton::WakeGenerator::Update\(\)](#).

22.19.3.7 bool TRITONAPI Triton::WakeGenerator::HasPropWash () const [inline]

Retrieves whether propeller backwash was enabled for this wake generator.

22.19.3.8 void TRITONAPI Triton::WakeGenerator::SetLODDistance (double *distance*)
[inline]

Sets the distance at which the number of visible wake waves will be halved.

For example, if a [WakeGenerator](#) has built up 8 wake waves and propeller wash segments and you pass a distance of 1000 meters, all 8 will be visible when the camera is closer than 1km to the WakeGenerator's position. Between 1 and 2km, every other wave will be skipped, resulting in 4 waves. From 3 to 4km, you'll get 2, which is as low as it will go.

Set this to 0 to disable LOD, which is the default.

22.19.3.9 void TRITONAPI Triton::WakeGenerator::SetParameters (const WakeGeneratorParameters & *parameters*)

Set this WakeGenerator's parameters using the [WakeGeneratorParameters](#) provided.

22.19.3.10 void TRITONAPI Triton::WakeGenerator::Update (const Vector3 & *pPosition*, const Vector3 & *pDirection*, double *pVelocity*, double *pTime*)

For any active [WakeGenerator](#), this should be called every frame to update its position and velocity.

No wake will be generated until this is called.

Parameters

<i>pPosition</i>	The position of the object generating the wake, such as the stern of a ship, in world coordinates.
<i>pDirection</i>	A normalized direction vector indicating the direction this object is moving in.
<i>pVelocity</i>	The velocity of the object generating the wake, in world units per second.
<i>pTime</i>	The current simulated time sample, in seconds. This may be relative to any reference point in time, as long as that reference point is consistent among the multiple calls to Update() .

The documentation for this class was generated from the following file:

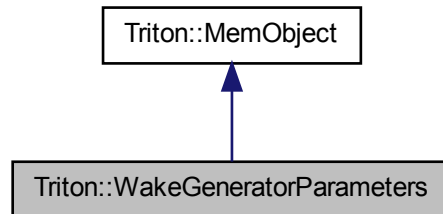
- C:/triton/trunk/Public Headers/[WakeGenerator.h](#)

22.20 Triton::WakeGeneratorParameters Class Reference

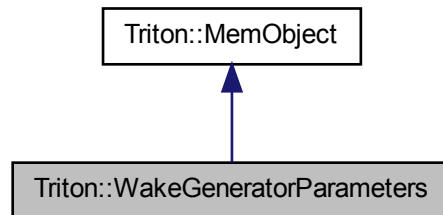
[WakeGeneratorParameters](#) contains the parameters required to construct a [Triton::WakeGenerator](#) object.

```
#include <WakeGenerator.h>
```

Inheritance diagram for Triton::WakeGeneratorParameters:



Collaboration diagram for Triton::WakeGeneratorParameters:



Public Member Functions

- [WakeGeneratorParameters\(\)](#)

The constructor populates all data members with reasonable default values, although you will at a minimum probably want to specify the `sprayEffects`, `length`, `beamWidth`, and `bowOffset`.

Public Attributes

- `bool` [sprayEffects](#)

Whether you wish this wake to emit spray particles originating from this wake generator.

- double [bowSprayOffset](#)
Use this to have spray particles emitted from a point different from the ship position.
- double [sprayVelocityScale](#)
A scaling factor for spray effects at the bow of the ship; this is applied to the initial velocity of the spray particles.
- double [spraySizeScale](#)
A scaling factor applied to the size of the bow spray particles and their random spread.
- double [bowWaveOffset](#)
Offset from the ship position, along the direction of travel, at which bow waves will originate.
- double [bowWaveScale](#)
A scaling factor for the bow wave's amplitude at the bow of the ship.
- double [bowWaveMax](#)
The maximum amplitude of the bow wave, or -1.0 for unbounded.
- double [bowSize](#)
The size of the bow in world units; affects the wavelength of the bow wave, and the initial spread of spray particles at the bow.
- double [sternWaveOffset](#)
Offset for the stern wakes.
- double [length](#)
The length of the object generating the wake.
- double [beamWidth](#)
The width of the object generating the wake.
- double [draft](#)
The "draft" of the ship, or the depth the hull extends to underwater.
- bool [propWash](#)
Whether you want propeller backwash effects generated from this wake generator.
- double [propWashOffset](#)
Use this to have propeller backwash effects generated from a point different from the ship position.
- double [propWashWidthMultiplier](#)
Multiplies the beam width by this value to arrive at the width of the prop wash effect.

- int [numHullSprays](#)
How many spray particle systems are emitted periodically along the hull of the ship.
- double [hullSprayStartOffset](#)
The offset at which hull spray effects begin.
- double [hullSprayEndOffset](#)
The offset at which hull spray effects end.
- double [hullSprayScale](#)
The initial velocity of hull spray particles as a percent of the ship velocity (0-1)
- double [hullSpraySizeScale](#)
A scaling factor applied to the hull spray particles.
- double [hullSprayVerticalOffset](#)
A vertical offset to the starting point of new hull spray particles.

22.20.1 Detailed Description

[WakeGeneratorParameters](#) contains the parameters required to construct a [Triton::WakeGenerator](#) object.

22.20.2 Constructor & Destructor Documentation

22.20.2.1 Triton::WakeGeneratorParameters::WakeGeneratorParameters ()

The constructor populates all data members with reasonable default values, although you will at a minimum probably want to specify the `sprayEffects`, `length`, `beamWidth`, and `bowOffset`.

22.20.3 Member Data Documentation

22.20.3.1 double Triton::WakeGeneratorParameters::bowSize

The size of the bow in world units; affects the wavelength of the bow wave, and the initial spread of spray particles at the bow.

For a pointy bow, use a value of 0. Generally you'd only use this for things like ferries or LCACs.

22.20.3.2 double Triton::WakeGeneratorParameters::bowSprayOffset

Use this to have spray particles emitted from a point different from the ship position.

This distance will be added to the current wake generator's position along the direction of travel. Unused if sprayEffects is false.

22.20.3.3 double Triton::WakeGeneratorParameters::bowWaveOffset

Offset from the ship position, along the direction of travel, at which bow waves will originate.

22.20.3.4 double Triton::WakeGeneratorParameters::draft

The "draft" of the ship, or the depth the hull extends to underwater.

This affects the size of the bow wake.

22.20.3.5 int Triton::WakeGeneratorParameters::numHullSprays

How many spray particle systems are emitted periodically along the hull of the ship.

Set to 0 to disable hull sprays.

22.20.3.6 double Triton::WakeGeneratorParameters::propWashOffset

Use this to have propeller backwash effects generated from a point different from the ship position.

22.20.3.7 double Triton::WakeGeneratorParameters::sprayVelocityScale

A scaling factor for spray effects at the bow of the ship; this is applied to the initial velocity of the spray particles.

The documentation for this class was generated from the following file:

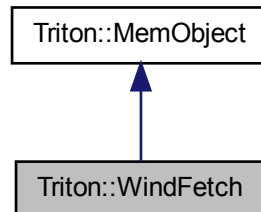
- C:/triton/trunk/Public Headers/[WakeGenerator.h](#)

22.21 Triton::WindFetch Class Reference

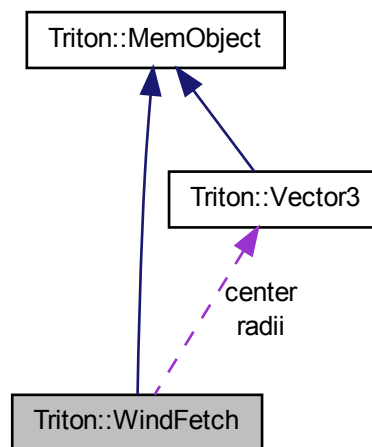
A localized or global area of wind of given speed and direction.

```
#include <WindFetch.h>
```

Inheritance diagram for Triton::WindFetch:



Collaboration diagram for Triton::WindFetch:



Public Member Functions

- [WindFetch](#) ()
Default constructor.
- void TRITONAPI [SetWind](#) (double speed, double direction)

Sets the wind speed and direction of this wind fetch.

- void TRITONAPI [SetLocalization](#) (const [Vector3](#) &center, const [Vector3](#) &radii)

Sets a localized area, in the form of an ellipsoid, in which the wind fetch is active.

- void TRITONAPI [SetFetchLength](#) (double fetch)

If using the JONSWAP model, swells will increase depending on the "fetch length," or the distance the wind has travelled.

- void TRITONAPI [ClearLocalization](#) ()

Clears any localization and makes this wind fetch globally applied.

- void TRITONAPI [ClearFetchLength](#) ()

Clears any explicit fetch length specified by [SetFetchLength\(\)](#).

- void TRITONAPI [GetWindAtLocation](#) (const [Vector3](#) &position, double &windSpeed, double &windDirection, double &fetchLength) const

Retrieves the wind direction and speed from this wind fetch at the given location.

22.21.1 Detailed Description

A localized or global area of wind of given speed and direction.

22.21.2 Constructor & Destructor Documentation

22.21.2.1 Triton::WindFetch::WindFetch ()

Default constructor.

22.21.3 Member Function Documentation

22.21.3.1 void TRITONAPI Triton::WindFetch::ClearFetchLength ()

Clears any explicit fetch length specified by [SetFetchLength\(\)](#).

22.21.3.2 void TRITONAPI Triton::WindFetch::ClearLocalization ()

Clears any localization and makes this wind fetch globally applied.

See also

[SetLocalization\(\)](#)

22.21.3.3 void TRITONAPI Triton::WindFetch::GetWindAtLocation (const Vector3 & *position*, double & *windSpeed*, double & *windDirection*, double & *fetchLength*) const

Retrieves the wind direction and speed from this wind fetch at the given location.

If the location specified is not included by the bounds of this wind fetch, or the wind fetch is not global, no wind will be returned.

Parameters

<i>position</i>	The location at which you want to retrieve wind information from this fetch.
<i>windSpeed</i>	The wind speed, in units per second, resulting from this fetch at the given position.
<i>windDirection</i>	The direction of the wind, in radians, resulting from this fetch at the given position. Represents the direction the wind is coming from, clockwise from North.
<i>fetchLength</i>	The distance the wind has travelled before reaching this position. Used only by the JONSWAP model. This will be based on distance from the fetch origin set with SetLocalization() , unless SetFetchLength() has been called which will take precedence. If neither SetLocalization() or SetFetchLength() has been called on this WindFetch , 0 will be returned and the JONSWAP model will switch to the PIERSON_MOSKOWITZ model.

22.21.3.4 void TRITONAPI Triton::WindFetch::SetFetchLength (double *fetch*)

If using the JONSWAP model, swells will increase depending on the "fetch length," or the distance the wind has travelled.

If you specified a fetch radius using [SetLocalization\(\)](#), this would normally be used to determine the fetch length. However, you may set an explicit fetch length (in world units) using this method, which will override [SetLocalization\(\)](#) until [ClearFetchLength\(\)](#) is called.

If you are not using the JONSWAP model, this value is ignored.

See also

[ClearFetchLength\(\)](#)

Parameters

<i>fetch</i>	The distance the wind has travelled before reaching the observer. Typical values are on the order of 100km.
--------------	---

22.21.3.5 void TRITONAPI Triton::WindFetch::SetLocalization (const Vector3 & *center*, const Vector3 & *radii*)

Sets a localized area, in the form of an ellipsoid, in which the wind fetch is active.

If this method is not called, the wind fetch is assumed to be global. If using the JON-

SWAP model, the size of the fetch specified will be used to determine the fetch length. It's important that this is realistic; fetch lengths on the order of 100km will produce the results you probably expect - so the distance from the observer to the center of the ellipsoid defined by this method should be on that order.

The center represents the 'origin' of the wind, and the wind will only affect observers within the ellipsoid defined by the radii given.

See also

[ClearLocalization\(\)](#)
[SetFetchLength\(\)](#)

Parameters

<i>center</i>	The center of the ellipsoid that models the bounds of the wind fetch.
<i>radii</i>	The radii in X, Y, and Z of the ellipsoid, in world units.

22.21.3.6 void TRITONAPI Triton::WindFetch::SetWind (double *speed*, double *direction*)

Sets the wind speed and direction of this wind fetch.

Parameters

<i>speed</i>	The wind speed in world units per second.
<i>direction</i>	The wind direction, in radians clockwise from North. Note this is the direction the wind is coming from - the waves will move in the opposite direction.

The documentation for this class was generated from the following file:

- C:/triton/trunk/Public Headers/[WindFetch.h](#)

Chapter 23

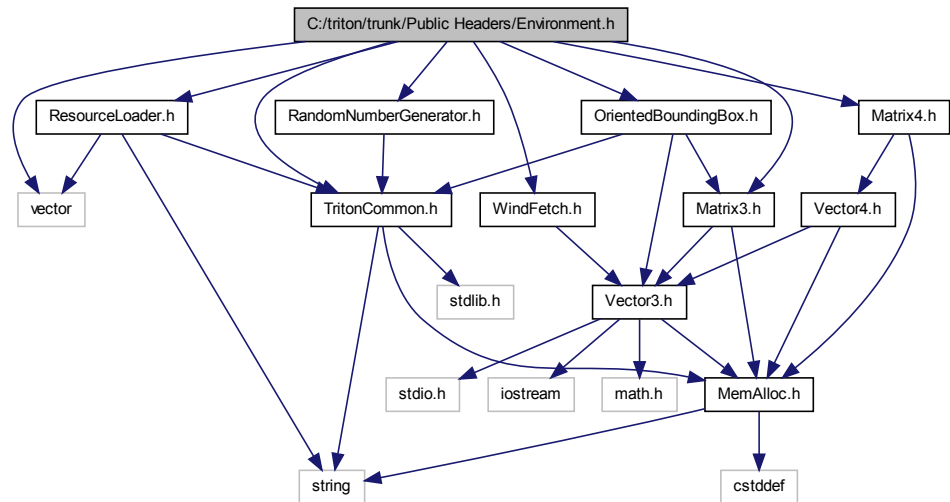
File Documentation

23.1 C:/triton/trunk/Public Headers/Environment.h File Reference

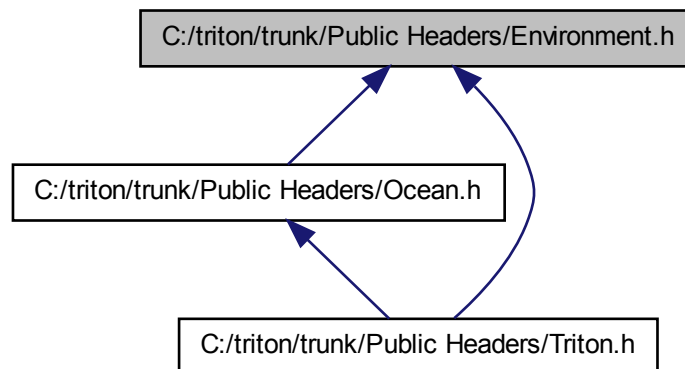
The public interface for setting Triton's environmental parameters.

```
#include "TritonCommon.h"
#include "ResourceLoader.h"
#include "RandomNumberGenerator.h"
#include "WindFetch.h"
#include "Matrix3.h"
#include "Matrix4.h"
#include "OrientedBoundingBox.h"
#include <vector>
```

Include dependency graph for Environment.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::BreakingWavesParameters](#)

Parameters to control behavior of breaking waves at shorelines, used by Environment::SetBreakingWaves().

- class [Triton::SwellDescription](#)

A structure containing a description of a swell in addition to local wind waves (from a distant storm perhaps.)

- class [Triton::Environment](#)

Triton's public interface for specifying the environmental conditions and camera properties.

Enumerations

- enum **CoordinateSystem** { ,

[Triton::WGS84_YUP](#), [Triton::SPHERICAL_ZUP](#), [Triton::SPHERICAL_YUP](#),
[Triton::FLAT_ZUP](#),
[Triton::FLAT_YUP](#) }

Supported coordinate systems for the Environment constructor.

- enum **Renderer** { ,

[Triton::OPENG_3_2](#), [Triton::OPENG_4_0](#), [Triton::OPENG_4_1](#), [Triton::DIRECTX_9](#),
[Triton::DIRECT3D9_EX](#), [Triton::DIRECTX_11](#), [Triton::NO_RENDERER](#) }

Support renderers for the Environment constructor.

- enum **EnvironmentError** { , [Triton::NO_CONFIG_FOUND](#), [Triton::NULL_RESOURCE_LOADER](#), [Triton::NO_DEVICE](#) }

Error codes returned from Environment::Initialize().

23.1.1 Detailed Description

The public interface for setting Triton's environmental parameters.

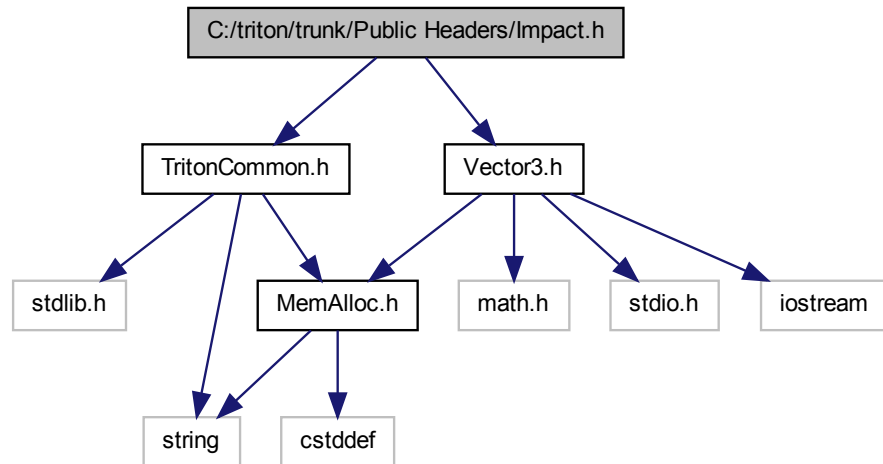
23.2 C:/triton/trunk/Public Headers/Impact.h File Reference

An object that generates impact wave and spray effects, ie from projectiles or explosions.

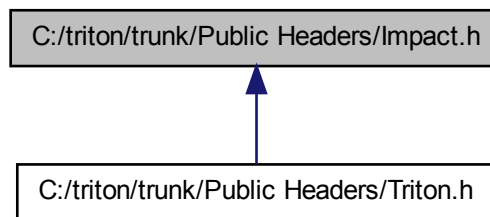
```
#include "TritonCommon.h"
```

```
#include "Vector3.h"
```

Include dependency graph for Impact.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Impact](#)

A [RotorWash](#) object will generate spray and circular waves on the ocean surface in the direction it is pointing.

23.2.1 Detailed Description

An object that generates impact wave and spray effects, ie from projectiles or explosions.

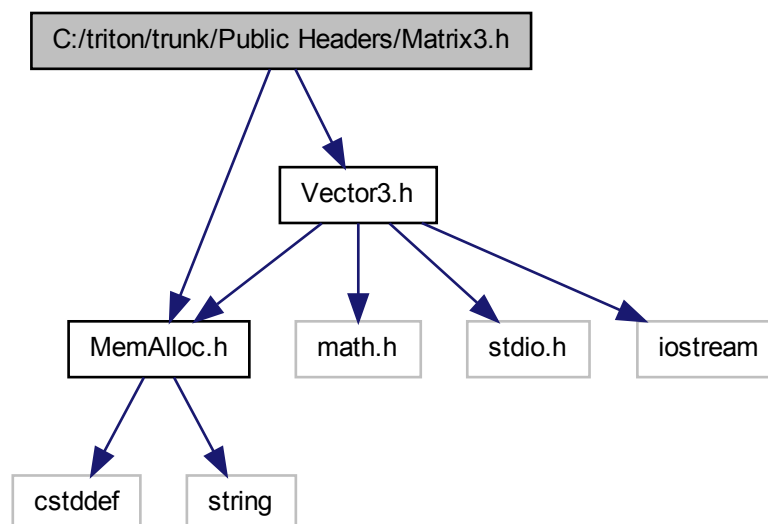
23.3 C:/triton/trunk/Public Headers/Matrix3.h File Reference

Implements a 3x3 matrix and its operations.

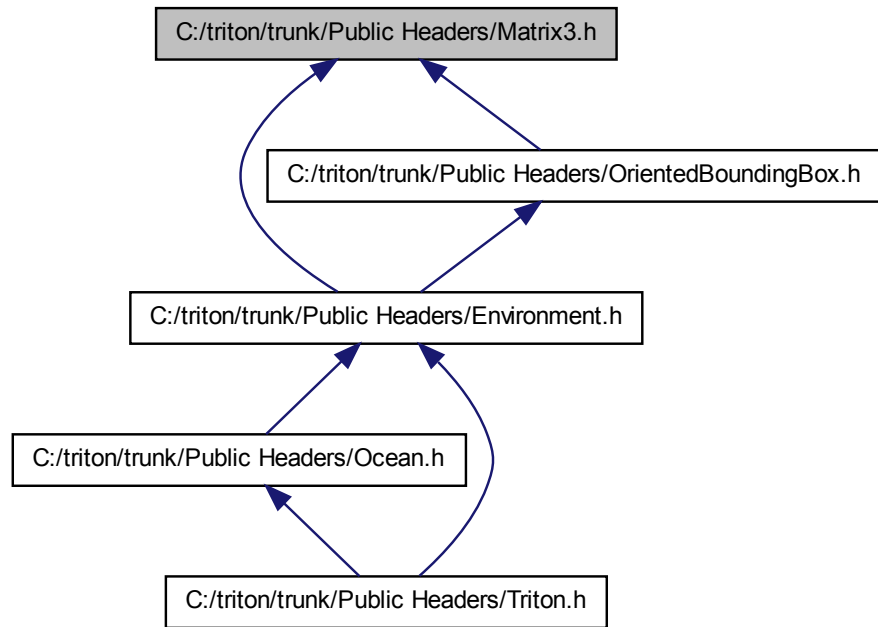
```
#include "MemAlloc.h"
```

```
#include "Vector3.h"
```

Include dependency graph for Matrix3.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Matrix3](#)

A simple 3x3 matrix class and its operations.

23.3.1 Detailed Description

Implements a 3x3 matrix and its operations.

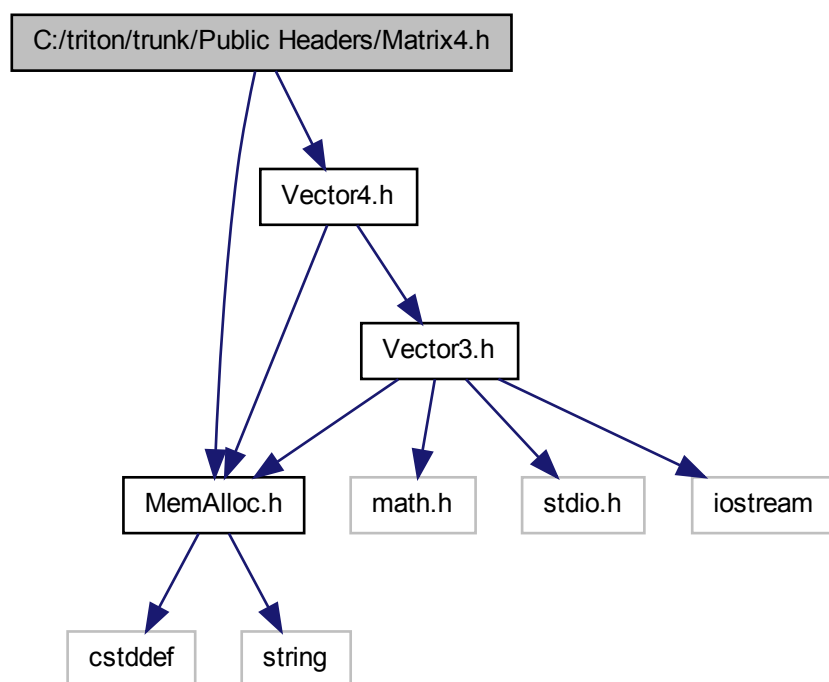
23.4 C:/triton/trunk/Public Headers/Matrix4.h File Reference

An implementation of a 4x4 matrix and some simple operations on it.

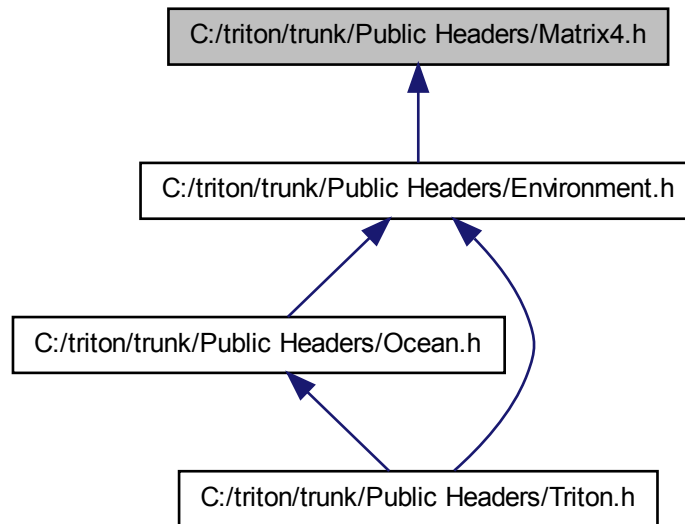
```
#include "MemAlloc.h"
```

```
#include "Vector4.h"
```

Include dependency graph for Matrix4.h:



This graph shows which files directly or indirectly include this file:



Classes

- class `Triton::Matrix4`

An implementation of a 4x4 matrix and some simple operations on it.

23.4.1 Detailed Description

An implementation of a 4x4 matrix and some simple operations on it.

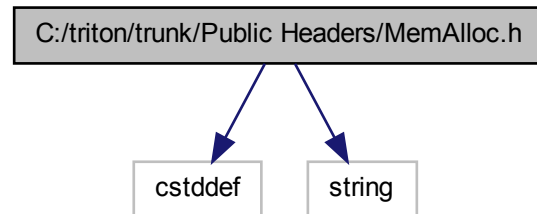
23.5 C:/triton/trunk/Public Headers/MemAlloc.h File Reference

Memory allocation interface for SilverLining.

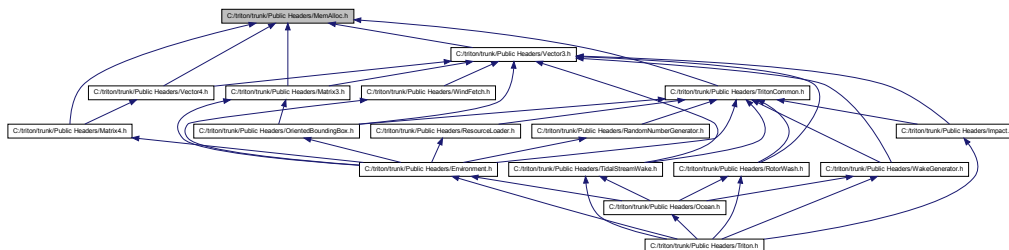
```
#include <stddef>
```

```
#include <string>
```

Include dependency graph for MemAlloc.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Allocator](#)

You may extend the [Allocator](#) class to hook your own memory management scheme into Triton.

- class [Triton::MemObject](#)

This base class for all Triton objects intercepts the new and delete operators, routing them through [Triton::Allocator\(\)](#).

23.5.1 Detailed Description

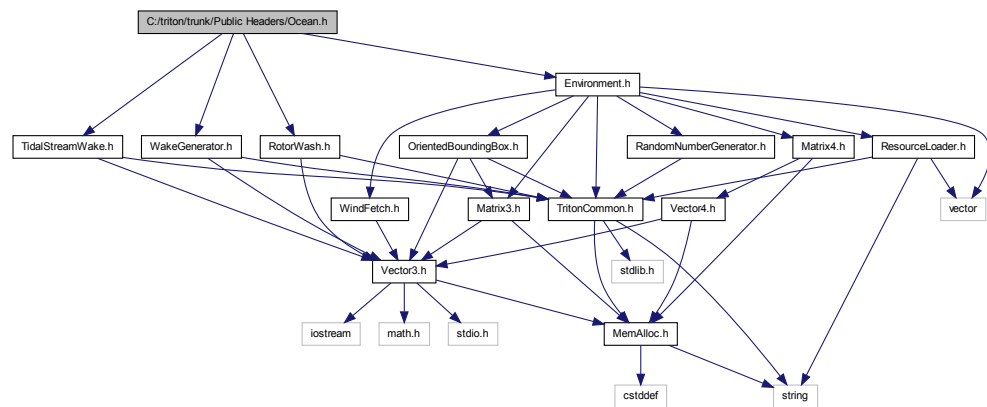
Memory allocation interface for SilverLining.

23.6 C:/triton/trunk/Public Headers/Ocean.h File Reference

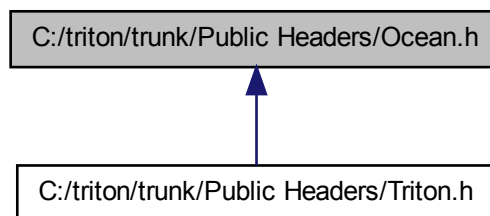
Triton's Ocean model interface.

```
#include "Environment.h"
#include "WakeGenerator.h"
#include "RotorWash.h"
#include "TidalStreamWake.h"
```

Include dependency graph for Ocean.h:



This graph shows which files directly or indirectly include this file:



23.7 C:/triton/trunk/Public Headers/OrientedBoundingBox.h File Reference 177

Classes

- class [Triton::Ocean](#)

The [Ocean](#) class allows you to configure and draw Triton's water simulation.

Enumerations

- enum **WaterModelTypes** { , [Triton::PIERSON_MOSKOWITZ](#), [Triton::JONSWAP](#) }

Enumerates the different water models available for simulating ocean waves.

- enum **Shaders**

Enumerates the different shader programs used internally by Triton.

- enum **OceanQuality**

Enumerates the ocean quality settings used in [Ocean::SetQuality\(\)](#)

23.6.1 Detailed Description

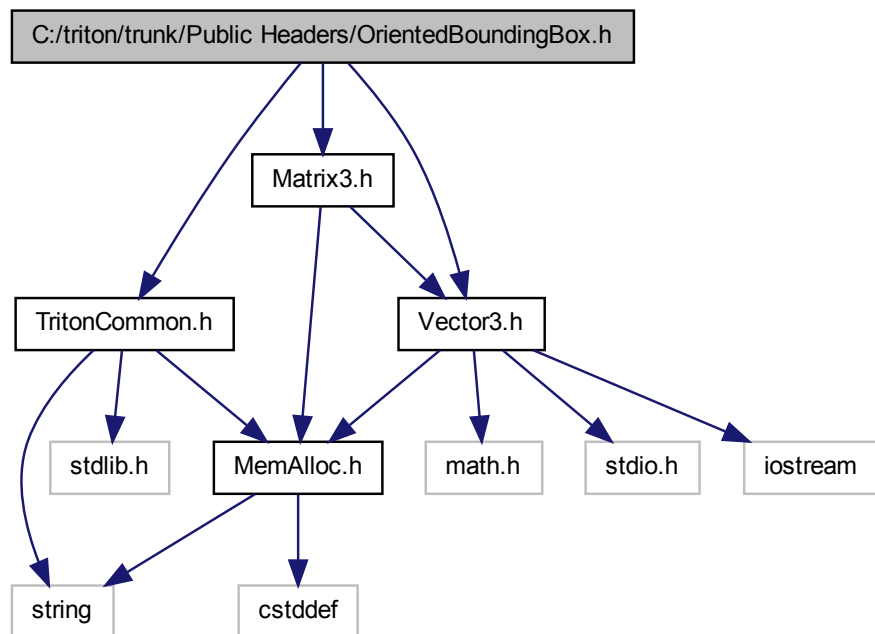
Triton's Ocean model interface.

23.7 C:/triton/trunk/Public Headers/OrientedBoundingBox.h File Reference

A class describing an oriented bounding box.

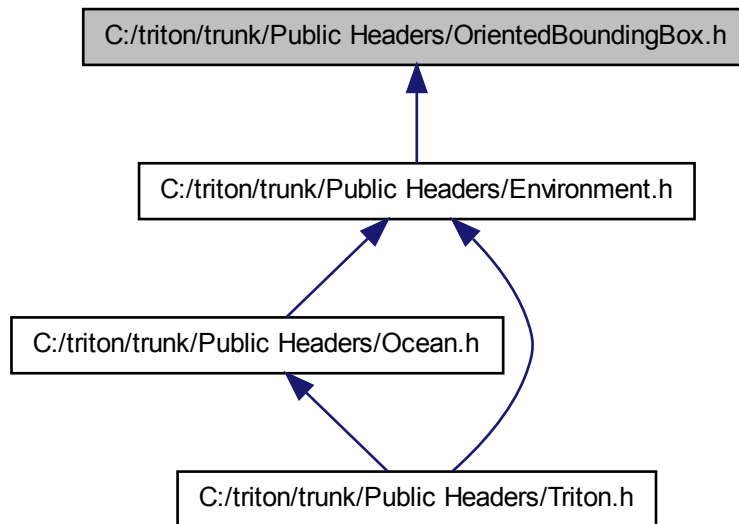
```
#include "TritonCommon.h"
#include "Vector3.h"
#include "Matrix3.h"
```

Include dependency graph for OrientedBoundingBox.h:



23.8 C:/triton/trunk/Public Headers/RandomNumberGenerator.h File Refered170

This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::OrientedBoundingBox](#)

An oriented bounding box defined by a center point and three axes.

23.7.1 Detailed Description

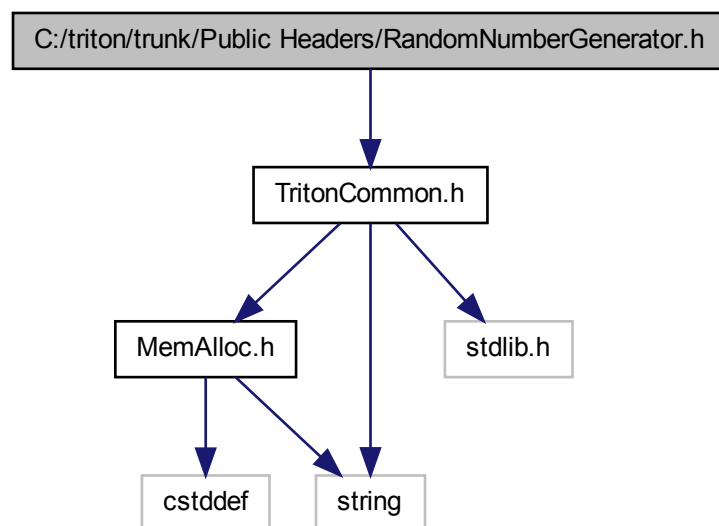
A class describing an oriented bounding box.

23.8 C:/triton/trunk/Public Headers/RandomNumberGenerator.h File Reference

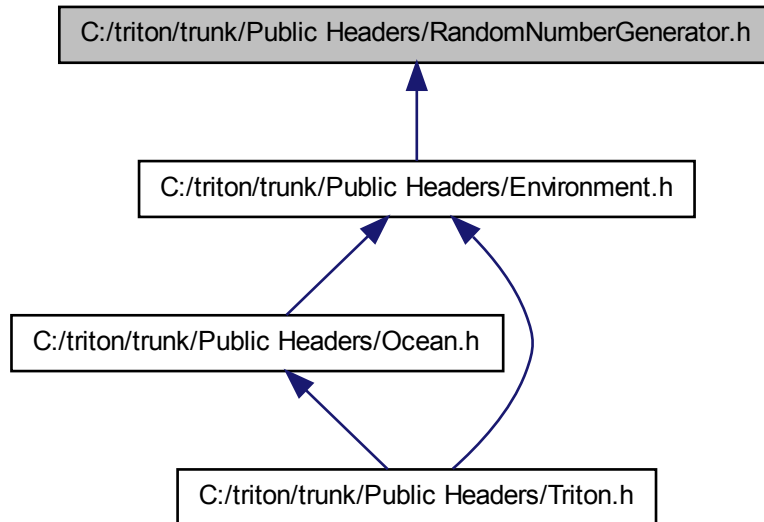
An interface for overriding Triton's generation of random numbers.

```
#include "TritonCommon.h"
```

Include dependency graph for RandomNumberGenerator.h:



This graph shows which files directly or indirectly include this file:



Classes

- class `Triton::RandomNumberGenerator`

An interface for generating random numbers in Triton.

23.8.1 Detailed Description

An interface for overriding Triton's generation of random numbers.

23.9 C:/triton/trunk/Public Headers/ResourceLoader.h File Reference

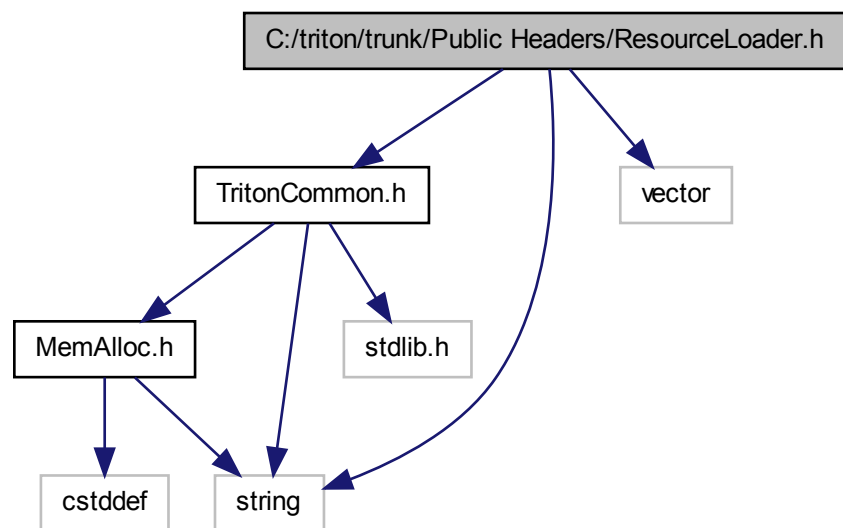
A class for loading Triton's resources from mass storage, which you may extend.

```
#include "TritonCommon.h"
```

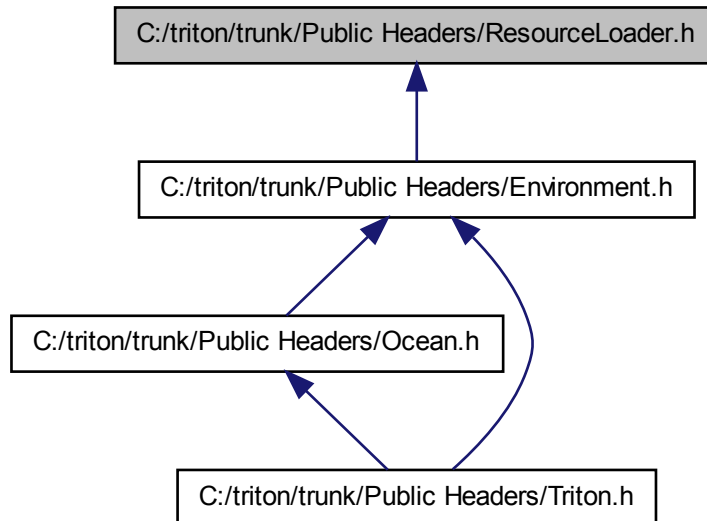
```
#include <vector>
```

```
#include <string>
```

Include dependency graph for ResourceLoader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::ResourceLoader](#)

This class is used whenever Triton needs to load textures, data files, or shaders from mass storage; you may extend this class to override our default use of POSIX filesystem calls with your own resource management if you wish.

23.9.1 Detailed Description

A class for loading Triton's resources from mass storage, which you may extend.

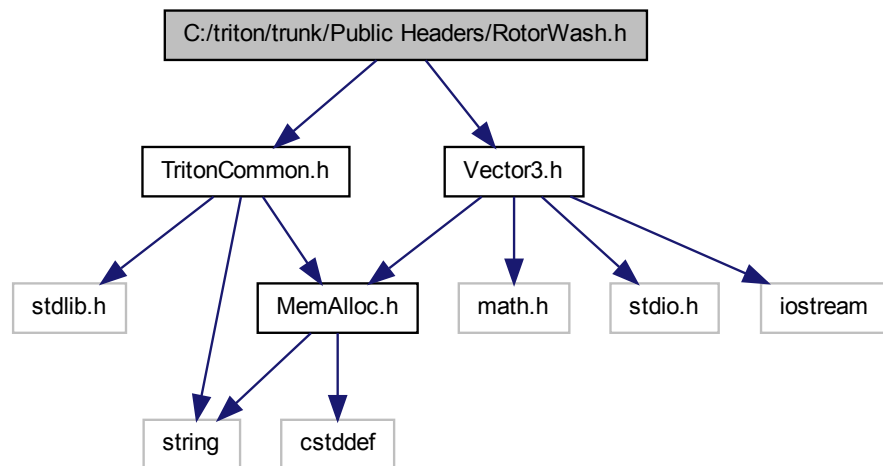
23.10 C:/triton/trunk/Public Headers/RotorWash.h File Reference

An object that generates rotor wash wave and spray effects.

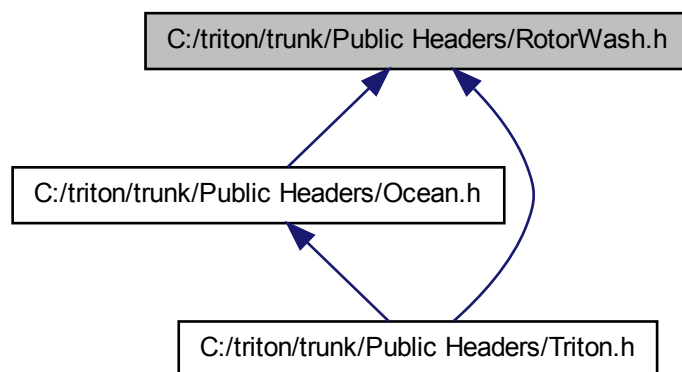
```
#include "TritonCommon.h"
```

```
#include "Vector3.h"
```

Include dependency graph for RotorWash.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::RotorWash](#)

A [RotorWash](#) object will generate spray and circular waves on the ocean surface in the direction it is pointing.

23.10.1 Detailed Description

An object that generates rotor wash wave and spray effects.

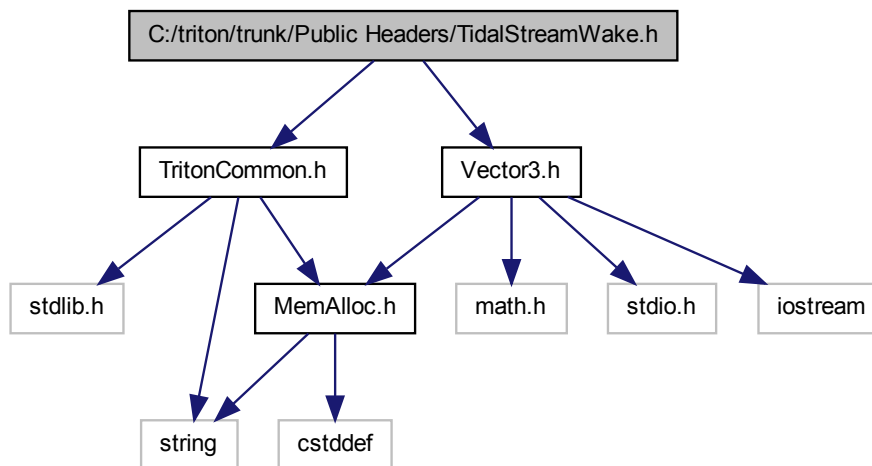
23.11 C:/triton/trunk/Public Headers/TidalStreamWake.h File Reference

An object that generates a static wake wave in a given direction, such as that generated by a buoy in a tidal stream.

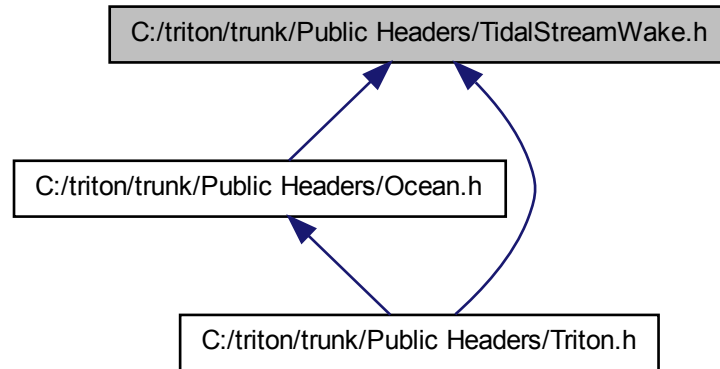
```
#include "TritonCommon.h"
```

```
#include "Vector3.h"
```

Include dependency graph for TidalStreamWake.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::TidalStreamWake](#)

An static wake pointing in a given direction at a fixed location, for example from a buoy or bridge pile in a current.

23.11.1 Detailed Description

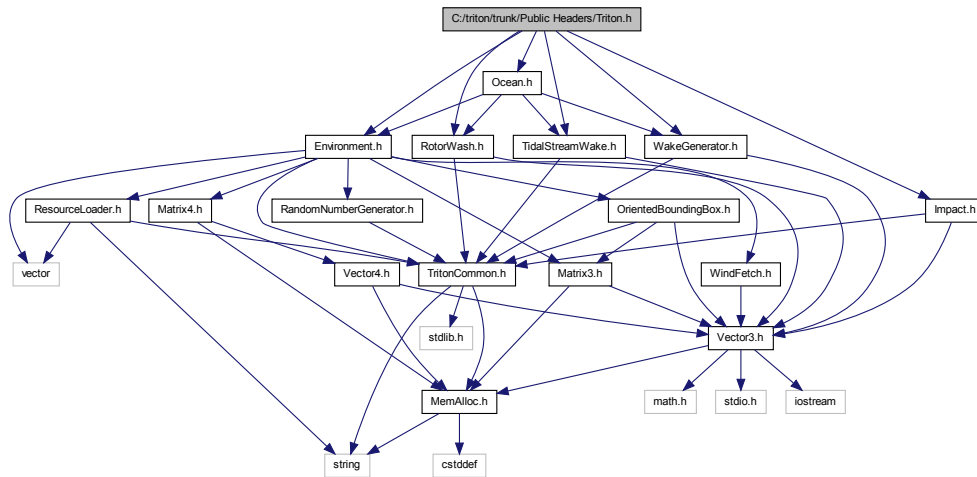
An object that generates a static wake wave in a given direction, such as that generated by a buoy in a tidal stream.

23.12 C:/triton/trunk/Public Headers/Triton.h File Reference

A convenience header that includes the main public headers for Triton.

```
#include "Environment.h"
#include "Ocean.h"
#include "WakeGenerator.h"
#include "RotorWash.h"
#include "Impact.h"
#include "TidalStreamWake.h"
```

Include dependency graph for Triton.h:



23.12.1 Detailed Description

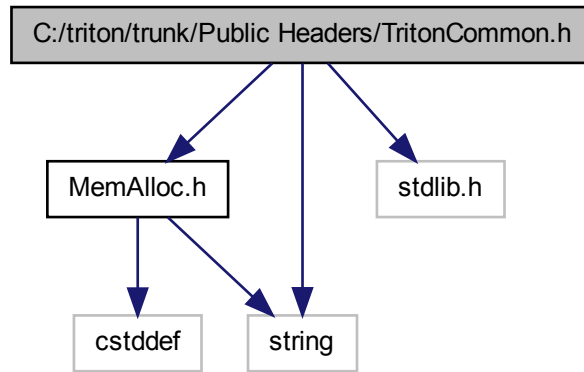
A convenience header that includes the main public headers for Triton.

23.13 C:/triton/trunk/Public Headers/TritonCommon.h File Reference

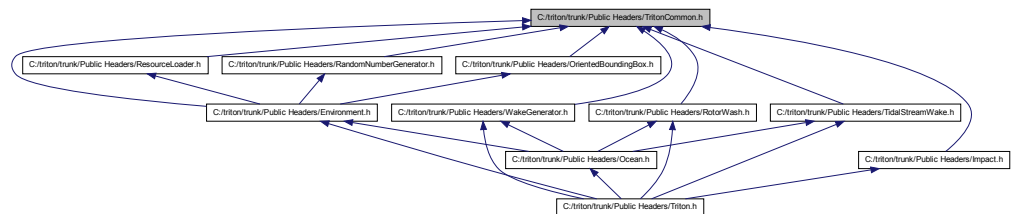
Common typedefs and defines used within Triton.

```
#include "MemAlloc.h"
#include <stdlib.h>
#include <string>
```

Include dependency graph for TritonCommon.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Utils](#)
A collection of static utility methods used by Triton.

Typedefs

- typedef int [Triton::ShaderHandle](#)
A renderer-agnostic handle for a shader.
- typedef int [Triton::TextureHandle](#)

A renderer-agnostic handle for a texture.

- typedef void * [Triton::DecalHandle](#)

A renderer-agnostic handle for a decal.

23.13.1 Detailed Description

Common typedefs and defines used within Triton.

23.14 C:/triton/trunk/Public Headers/Vector3.h File Reference

A 3D Vector class and its operations.

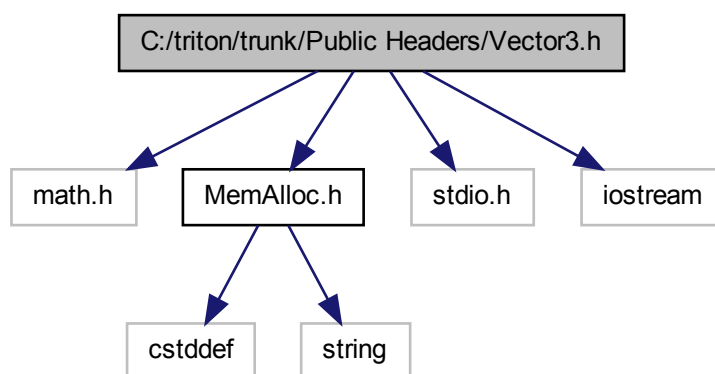
```
#include <math.h>
```

```
#include "MemAlloc.h"
```

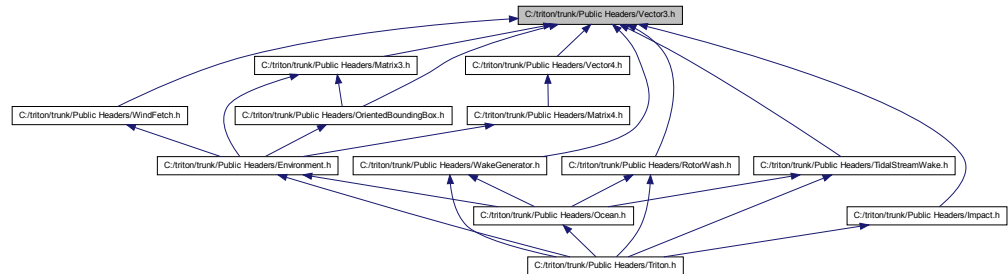
```
#include <stdio.h>
```

```
#include <iostream>
```

Include dependency graph for Vector3.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Vector3](#)

A 3D double-precision Vector class and its operations.

- class [Triton::Vector3f](#)

A 3D single-precision vector class, and its operations.

23.14.1 Detailed Description

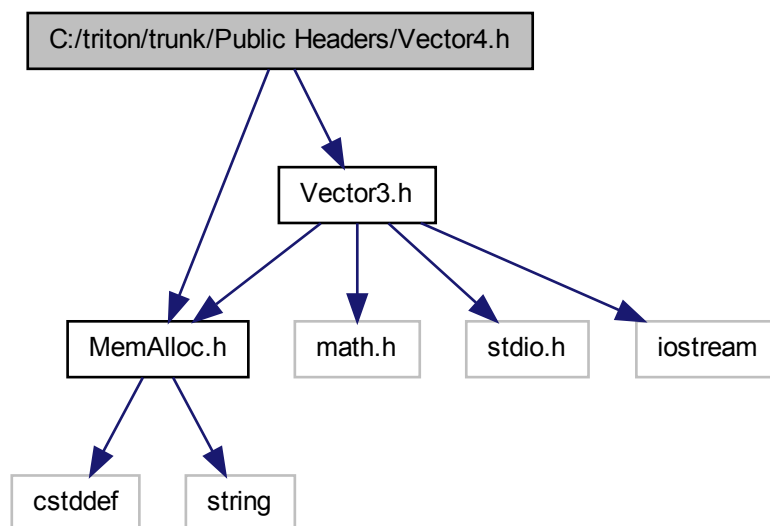
A 3D Vector class and its operations.

23.15 C:/triton/trunk/Public Headers/Vector4.h File Reference

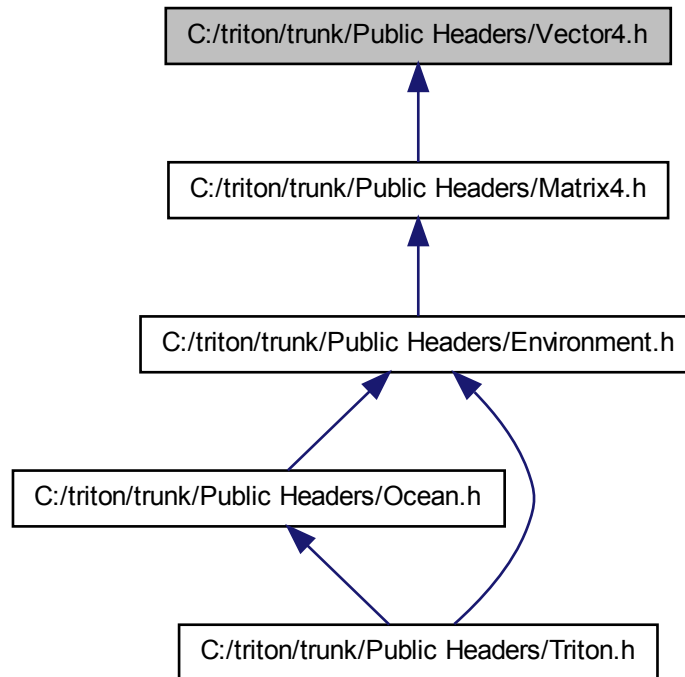
A simple 4D vector class.

```
#include "MemAlloc.h"
#include "Vector3.h"
```

Include dependency graph for Vector4.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::Vector4](#)

A simple double-precision 4D vector class with no operations defined.

23.15.1 Detailed Description

A simple 4D vector class.

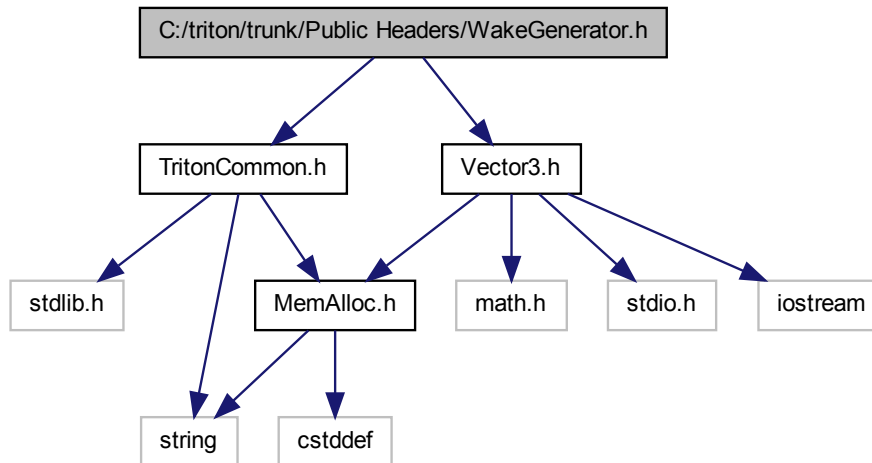
23.16 C:/triton/trunk/Public Headers/WakeGenerator.h File Reference

An object that generates a ship Kelvin wake as it moves.

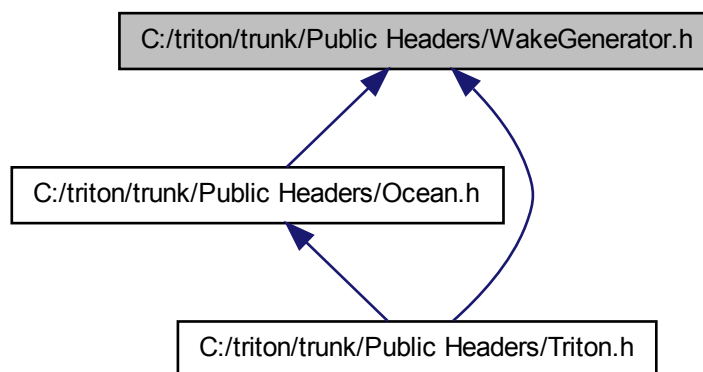
```
#include "TritonCommon.h"
```

```
#include "Vector3.h"
```

Include dependency graph for WakeGenerator.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::WakeGeneratorParameters](#)

[WakeGeneratorParameters](#) contains the parameters required to construct a [Triton::WakeGenerator](#) object.

- class [Triton::WakeGenerator](#)

A [WakeGenerator](#) represents an object on the water that generates a wake as it moves, such as a ship.

23.16.1 Detailed Description

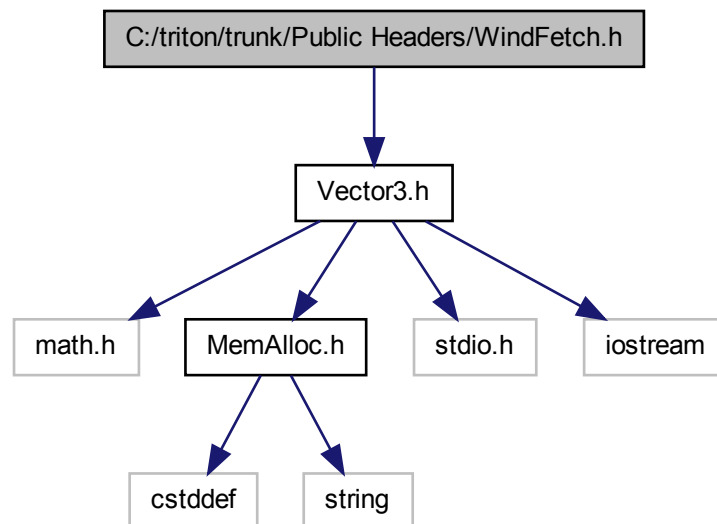
An object that generates a ship Kelvin wake as it moves.

23.17 C:/triton/trunk/Public Headers/WindFetch.h File Reference

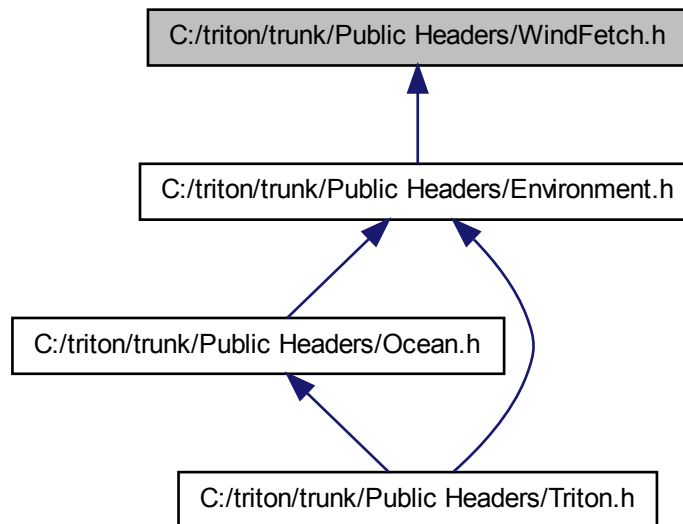
A localized or global area of wind of given speed and direction.

```
#include "Vector3.h"
```

Include dependency graph for WindFetch.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Triton::WindFetch](#)

A localized or global area of wind of given speed and direction.

23.17.1 Detailed Description

A localized or global area of wind of given speed and direction.

Index

~Environment
 Triton::Environment, 78
~Matrix3
 Triton::Matrix3, 98
~Matrix4
 Triton::Matrix4, 102
~Ocean
 Triton::Ocean, 110
AddSwell
 Triton::Environment, 78
AddWindFetch
 Triton::Environment, 78
bowSize
 Triton::WakeGeneratorParameters, 150
bowSprayOffset
 Triton::WakeGeneratorParameters, 151
bowWaveOffset
 Triton::WakeGeneratorParameters, 151
BreakingWavesParameters
 Triton::BreakingWavesParameters, 70
C:/triton/trunk/Public Headers/Environment.h, 157
C:/triton/trunk/Public Headers/Impact.h, 159
C:/triton/trunk/Public Headers/Matrix3.h, 161
C:/triton/trunk/Public Headers/Matrix4.h, 162
C:/triton/trunk/Public Headers/MemAlloc.h, 164
C:/triton/trunk/Public Headers/Ocean.h, 166
C:/triton/trunk/Public Headers/RandomNumberGenerator.h, 167
C:/triton/trunk/Public Headers/ResourceLoader.h, 169
C:/triton/trunk/Public Headers/RotorWash.h, 171
C:/triton/trunk/Public Headers/Triton.h, 173
C:/triton/trunk/Public Headers/TritonCommon.h, 174
C:/triton/trunk/Public Headers/Vector3.h, 175
C:/triton/trunk/Public Headers/Vector4.h, 177
C:/triton/trunk/Public Headers/WakeGenerator.h, 178
C:/triton/trunk/Public Headers/WindFetch.h, 180
ClearFetchLength
 Triton::WindFetch, 153
ClearGLErrors
 Triton::Utils, 132
ClearLocalization
 Triton::WindFetch, 153
ClearSwells
 Triton::Environment, 79
ClearWakes
 Triton::WakeGenerator, 146
ClearWindFetches
 Triton::Environment, 79
ComputeReflectionMatrices
 Triton::Ocean, 110
Create
 Triton::Ocean, 111
Cross
 Triton::Vector3, 136
CullSphere
 Triton::Environment, 79
D3D9DeviceLost
 Triton::Ocean, 112
D3D9DeviceReset
 Triton::Ocean, 112
Dot
 Triton::Vector3, 136
 Triton::Vector3f, 140
 Triton::Vector4, 143
draft
 Triton::WakeGeneratorParameters, 151
Draw
 Triton::Ocean, 112
EnableOpenMP

- Triton::Environment, [79](#)
- EnableSpray
 - Triton::Ocean, [113](#)
- EnableWireframe
 - Triton::Ocean, [113](#)
- Environment
 - Triton::Environment, [78](#)
- FreeResource
 - Triton::ResourceLoader, [125](#)
- FromRx
 - Triton::Matrix3, [99](#)
- FromRy
 - Triton::Matrix3, [99](#)
- FromRz
 - Triton::Matrix3, [99](#)
- FromXYZ
 - Triton::Matrix3, [99](#)
- GetAboveWaterVisibility
 - Triton::Environment, [80](#)
- GetAmbientLightColor
 - Triton::Environment, [80](#)
- GetBelowWaterVisibility
 - Triton::Environment, [80](#)
- GetBreakingWavesParameters
 - Triton::Environment, [80](#)
- GetCameraMatrix
 - Triton::Environment, [80](#)
- GetCameraPosition
 - Triton::Environment, [81](#)
- GetChoppiness
 - Triton::Ocean, [113](#)
- GetConfigOption
 - Triton::Environment, [81](#)
- GetCoordinateSystem
 - Triton::Environment, [81](#)
- GetDepth
 - Triton::Ocean, [113](#)
- GetDepthOffset
 - Triton::Ocean, [114](#)
- GetDevice
 - Triton::Environment, [81](#)
- GetDirectionalLightColor
 - Triton::Environment, [81](#)
- GetDisplacementDampingDistance
 - Triton::Ocean, [114](#)
- GetDLLExtension
 - Triton::Utils, [132](#)
- GetDLLPath
 - Triton::Utils, [132](#)
- GetDLLSuffix
 - Triton::Utils, [132](#)
- GetDX9Macros
 - Triton::Utils, [132](#)
- GetEnvironment
 - Triton::Ocean, [114](#)
- GetEnvironmentMap
 - Triton::Environment, [81](#)
- GetEnvironmentMapMatrix
 - Triton::Environment, [82](#)
- GetFFTName
 - Triton::Ocean, [114](#)
- GetHeight
 - Triton::Ocean, [114](#)
- GetHeightMap
 - Triton::Environment, [82](#)
- GetHeightMapMatrix
 - Triton::Environment, [82](#)
- GetIntersection
 - Triton::Ocean, [115](#)
- GetLightDirection
 - Triton::Environment, [82](#)
- GetLODDistance
 - Triton::WakeGenerator, [146](#)
- GetLoopingPeriod
 - Triton::Ocean, [115](#)
- GetNumTriangles
 - Triton::Ocean, [116](#)
- GetOpenMPEnabled
 - Triton::Environment, [82](#)
- GetParameters
 - Triton::WakeGenerator, [146](#)
- GetParticleShaderFileName
 - Triton::Utils, [132](#)
- GetPlanarReflectionBlend
 - Triton::Ocean, [116](#)
- GetPlanarReflectionDisplacementScale
 - Triton::Environment, [82](#)
- GetPlanarReflectionMap
 - Triton::Environment, [83](#)
- GetPlanarReflectionMapMatrix
 - Triton::Environment, [83](#)
- GetPosition
 - Triton::Impact, [95](#)
 - Triton::RotorWash, [128](#)
 - Triton::WakeGenerator, [146](#)
- GetProjectionMatrix
 - Triton::Environment, [83](#)
- GetRandomDouble

- Triton::RandomNumberGenerator, 122
- GetRandomInt
 - Triton::RandomNumberGenerator, 123
- GetRandomNumberGenerator
 - Triton::Environment, 83
- GetRefractionColor
 - Triton::Ocean, 116
- GetRenderer
 - Triton::Environment, 83
- GetResourceLoader
 - Triton::Environment, 83
- GetRightVector
 - Triton::Environment, 83
- GetRow
 - Triton::Matrix4, 103
- GetSeaLevel
 - Triton::Environment, 84
- GetShaderObject
 - Triton::Ocean, 116
- GetSternPosition
 - Triton::WakeGenerator, 146
- GetUpVector
 - Triton::Environment, 84
- GetVelocity
 - Triton::Impact, 95
 - Triton::RotorWash, 128
 - Triton::WakeGenerator, 146
- GetWaterShaderFileName
 - Triton::Utils, 132
- GetWaveHeading
 - Triton::Ocean, 116
- GetWind
 - Triton::Environment, 84
- GetWindAtLocation
 - Triton::WindFetch, 153
- GetWorldUnits
 - Triton::Environment, 84
- HasPropWash
 - Triton::WakeGenerator, 147
- Impact
 - Triton::Impact, 95
- Initialize
 - Triton::Environment, 85
- InverseCramers
 - Triton::Matrix4, 103
- IsCameraAboveWater
 - Triton::Ocean, 117
- IsDirectX
 - Triton::Environment, 85
- IsGeocentric
 - Triton::Environment, 85
- IsOpenGL
 - Triton::Environment, 86
- Length
 - Triton::Vector3, 136
 - Triton::Vector3f, 140
- LoadResource
 - Triton::ResourceLoader, 125
- Matrix3
 - Triton::Matrix3, 98
- Matrix4
 - Triton::Matrix4, 102
- Normalize
 - Triton::Vector3, 136
 - Triton::Vector3f, 140
- numHullSprays
 - Triton::WakeGeneratorParameters, 151
- operator*
 - Triton::Matrix3, 99, 100
 - Triton::Matrix4, 103
 - Triton::Vector3, 136
 - Triton::Vector3f, 140
 - Triton::Vector4, 143
- operator+
 - Triton::Vector3, 136
 - Triton::Vector3f, 140
 - Triton::Vector4, 143
- operator-
 - Triton::Vector3, 137
 - Triton::Vector3f, 140
 - Triton::Vector4, 143
- operator==
 - Triton::Vector3, 137
- PrintGLErrors
 - Triton::Utils, 133
- propWashOffset
 - Triton::WakeGeneratorParameters, 151
- ResourceLoader
 - Triton::ResourceLoader, 125
- RotorWash
 - Triton::RotorWash, 128
- Serialize

- Triton::Vector3, 137
- SetAboveWaterVisibility
 - Triton::Environment, 86
- SetAllocator
 - Triton::Allocator, 68
- SetAmbientLight
 - Triton::Environment, 86
- SetAmplitude
 - Triton::BreakingWavesParameters, 70
- SetBelowWaterVisibility
 - Triton::Environment, 86
- SetBreakingWavesParameters
 - Triton::Environment, 87
- SetCameraMatrix
 - Triton::Environment, 87
- SetChoppiness
 - Triton::Ocean, 117
- SetConfigOption
 - Triton::Environment, 87
- SetDepth
 - Triton::Ocean, 117
- SetDepthOffset
 - Triton::Ocean, 117
- SetDirectionalLight
 - Triton::Environment, 87
- SetDisplacementDampingDistance
 - Triton::Ocean, 118
- SetDouglasSeaScale
 - Triton::Environment, 88
- SetEnvironmentMap
 - Triton::Environment, 88
- SetFetchLength
 - Triton::WindFetch, 154
- SetHeightMap
 - Triton::Environment, 88
- SetLicenseCode
 - Triton::Environment, 89
- SetLocalization
 - Triton::WindFetch, 154
- SetLODDistance
 - Triton::WakeGenerator, 147
- SetLoopingPeriod
 - Triton::Ocean, 118
- SetParameters
 - Triton::WakeGenerator, 147
- SetPatchShader
 - Triton::Ocean, 118
- SetPlanarReflectionBlend
 - Triton::Ocean, 119
- SetPlanarReflectionMap
 - Triton::Environment, 90
- SetProjectionMatrix
 - Triton::Environment, 91
- SetRandomNumberGenerator
 - Triton::Environment, 91
- SetRandomSeed
 - Triton::RandomNumberGenerator, 123
- SetRefractionColor
 - Triton::Ocean, 119
- SetResourceDirPath
 - Triton::ResourceLoader, 126
- SetSeaLevel
 - Triton::Environment, 91
- SetSteepness
 - Triton::BreakingWavesParameters, 70
- SetSteepnessVariance
 - Triton::BreakingWavesParameters, 71
- SetSurgeDepth
 - Triton::BreakingWavesParameters, 71
- SetWaveDirection
 - Triton::BreakingWavesParameters, 71
- SetWavelength
 - Triton::BreakingWavesParameters, 71
- SetWavelengthVariance
 - Triton::BreakingWavesParameters, 71
- SetWind
 - Triton::WindFetch, 155
- SetWorldUnits
 - Triton::Environment, 92
- SimulateSeaState
 - Triton::Environment, 92
- SprayEnabled
 - Triton::Ocean, 120
- sprayVelocityScale
 - Triton::WakeGeneratorParameters, 151
- SquaredLength
 - Triton::Vector3, 137
- ToFloatArray
 - Triton::Matrix3, 99
 - Triton::Matrix4, 103
- Transpose
 - Triton::Matrix3, 99
 - Triton::Matrix4, 103
- Trigger
 - Triton::Impact, 95
- Triton::Allocator, 67
 - SetAllocator, 68
- Triton::BreakingWavesParameters, 68
 - BreakingWavesParameters, 70

- SetAmplitude, [70](#)
- SetSteepness, [70](#)
- SetSteepnessVariance, [71](#)
- SetSurgeDepth, [71](#)
- SetWaveDirection, [71](#)
- SetWavelength, [71](#)
- SetWavelengthVariance, [71](#)
- Triton::Environment, [72](#)
 - ~Environment, [78](#)
 - AddSwell, [78](#)
 - AddWindFetch, [78](#)
 - ClearSwells, [79](#)
 - ClearWindFetches, [79](#)
 - CullSphere, [79](#)
 - EnableOpenMP, [79](#)
 - Environment, [78](#)
 - GetAboveWaterVisibility, [80](#)
 - GetAmbientLightColor, [80](#)
 - GetBelowWaterVisibility, [80](#)
 - GetBreakingWavesParameters, [80](#)
 - GetCameraMatrix, [80](#)
 - GetCameraPosition, [81](#)
 - GetConfigOption, [81](#)
 - GetCoordinateSystem, [81](#)
 - GetDevice, [81](#)
 - GetDirectionalLightColor, [81](#)
 - GetEnvironmentMap, [81](#)
 - GetEnvironmentMapMatrix, [82](#)
 - GetHeightMap, [82](#)
 - GetHeightMapMatrix, [82](#)
 - GetLightDirection, [82](#)
 - GetOpenMPEnabled, [82](#)
 - GetPlanarReflectionDisplacementScale, [82](#)
 - GetPlanarReflectionMap, [83](#)
 - GetPlanarReflectionMapMatrix, [83](#)
 - GetProjectionMatrix, [83](#)
 - GetRandomNumberGenerator, [83](#)
 - GetRenderer, [83](#)
 - GetResourceLoader, [83](#)
 - GetRightVector, [83](#)
 - GetSeaLevel, [84](#)
 - GetUpVector, [84](#)
 - GetWind, [84](#)
 - GetWorldUnits, [84](#)
 - Initialize, [85](#)
 - IsDirectX, [85](#)
 - IsGeocentric, [85](#)
 - IsOpenGL, [86](#)
 - SetAboveWaterVisibility, [86](#)
 - SetAmbientLight, [86](#)
 - SetBelowWaterVisibility, [86](#)
 - SetBreakingWavesParameters, [87](#)
 - SetCameraMatrix, [87](#)
 - SetConfigOption, [87](#)
 - SetDirectionalLight, [87](#)
 - SetDouglasSeaScale, [88](#)
 - SetEnvironmentMap, [88](#)
 - SetHeightMap, [88](#)
 - SetLicenseCode, [89](#)
 - SetPlanarReflectionMap, [90](#)
 - SetProjectionMatrix, [91](#)
 - SetRandomNumberGenerator, [91](#)
 - SetSeaLevel, [91](#)
 - SetWorldUnits, [92](#)
 - SimulateSeaState, [92](#)
 - TRITON_VECTOR, [92](#)
- Triton::Impact, [93](#)
 - GetPosition, [95](#)
 - GetVelocity, [95](#)
 - Impact, [95](#)
 - Trigger, [95](#)
- Triton::Matrix3, [96](#)
 - ~Matrix3, [98](#)
 - FromRx, [99](#)
 - FromRy, [99](#)
 - FromRz, [99](#)
 - FromXYZ, [99](#)
 - Matrix3, [98](#)
 - operator*, [99](#), [100](#)
 - ToFloatArray, [99](#)
 - Transpose, [99](#)
- Triton::Matrix4, [100](#)
 - ~Matrix4, [102](#)
 - GetRow, [103](#)
 - InverseCramers, [103](#)
 - Matrix4, [102](#)
 - operator*, [103](#)
 - ToFloatArray, [103](#)
 - Transpose, [103](#)
- Triton::MemObject, [104](#)
- Triton::Ocean, [106](#)
 - ~Ocean, [110](#)
 - ComputeReflectionMatrices, [110](#)
 - Create, [111](#)
 - D3D9DeviceLost, [112](#)
 - D3D9DeviceReset, [112](#)
 - Draw, [112](#)
 - EnableSpray, [113](#)
 - EnableWireframe, [113](#)

- GetChoppiness, [113](#)
- GetDepth, [113](#)
- GetDepthOffset, [114](#)
- GetDisplacementDampingDistance, [114](#)
- GetEnvironment, [114](#)
- GetFFTName, [114](#)
- GetHeight, [114](#)
- GetIntersection, [115](#)
- GetLoopingPeriod, [115](#)
- GetNumTriangles, [116](#)
- GetPlanarReflectionBlend, [116](#)
- GetRefractionColor, [116](#)
- GetShaderObject, [116](#)
- GetWaveHeading, [116](#)
- IsCameraAboveWater, [117](#)
- SetChoppiness, [117](#)
- SetDepth, [117](#)
- SetDepthOffset, [117](#)
- SetDisplacementDampingDistance, [118](#)
- SetLoopingPeriod, [118](#)
- SetPatchShader, [118](#)
- SetPlanarReflectionBlend, [119](#)
- SetRefractionColor, [119](#)
- SprayEnabled, [120](#)
- UnsetPatchShader, [120](#)
- UpdateSimulation, [120](#)
- Triton::RandomNumberGenerator, [121](#)
 - GetRandomDouble, [122](#)
 - GetRandomInt, [123](#)
 - SetRandomSeed, [123](#)
- Triton::ResourceLoader, [123](#)
 - FreeResource, [125](#)
 - LoadResource, [125](#)
 - ResourceLoader, [125](#)
 - SetResourceDirPath, [126](#)
- Triton::RotorWash, [127](#)
 - GetPosition, [128](#)
 - GetVelocity, [128](#)
 - RotorWash, [128](#)
 - Update, [129](#)
- Triton::SwellDescription, [129](#)
- Triton::Utils, [131](#)
 - ClearGLErrors, [132](#)
 - GetDLLExtension, [132](#)
 - GetDLLPath, [132](#)
 - GetDLLSuffix, [132](#)
 - GetDX9Macros, [132](#)
 - GetParticleShaderFileName, [132](#)
 - GetWaterShaderFileName, [132](#)
 - PrintGLErrors, [133](#)
- Triton::Vector3, [133](#)
 - Cross, [136](#)
 - Dot, [136](#)
 - Length, [136](#)
 - Normalize, [136](#)
 - operator*, [136](#)
 - operator+, [136](#)
 - operator-, [137](#)
 - operator==, [137](#)
 - Serialize, [137](#)
 - SquaredLength, [137](#)
 - Unserialize, [137](#)
 - Vector3, [135](#)
 - x, [137](#)
- Triton::Vector3f, [137](#)
 - Dot, [140](#)
 - Length, [140](#)
 - Normalize, [140](#)
 - operator*, [140](#)
 - operator+, [140](#)
 - operator-, [140](#)
 - Vector3f, [139](#)
 - x, [140](#)
- Triton::Vector4, [141](#)
 - Dot, [143](#)
 - operator*, [143](#)
 - operator+, [143](#)
 - operator-, [143](#)
 - Vector4, [142](#)
 - x, [143](#)
- Triton::WakeGenerator, [143](#)
 - ClearWakes, [146](#)
 - GetLODDistance, [146](#)
 - GetParameters, [146](#)
 - GetPosition, [146](#)
 - GetSternPosition, [146](#)
 - GetVelocity, [146](#)
 - HasPropWash, [147](#)
 - SetLODDistance, [147](#)
 - SetParameters, [147](#)
 - Update, [147](#)
 - WakeGenerator, [146](#)
- Triton::WakeGeneratorParameters, [148](#)
 - bowSize, [150](#)
 - bowSprayOffset, [151](#)
 - bowWaveOffset, [151](#)
 - draft, [151](#)
 - numHullSprays, [151](#)
 - propWashOffset, [151](#)
 - sprayVelocityScale, [151](#)

- WakeGeneratorParameters, [150](#)
- Triton::WindFetch, [151](#)
 - ClearFetchLength, [153](#)
 - ClearLocalization, [153](#)
 - GetWindAtLocation, [153](#)
 - SetFetchLength, [154](#)
 - SetLocalization, [154](#)
 - SetWind, [155](#)
 - WindFetch, [153](#)
- TRITON_VECTOR
 - Triton::Environment, [92](#)
- Unserialize
 - Triton::Vector3, [137](#)
- UnsetPatchShader
 - Triton::Ocean, [120](#)
- Update
 - Triton::RotorWash, [129](#)
 - Triton::WakeGenerator, [147](#)
- UpdateSimulation
 - Triton::Ocean, [120](#)
- Vector3
 - Triton::Vector3, [135](#)
- Vector3f
 - Triton::Vector3f, [139](#)
- Vector4
 - Triton::Vector4, [142](#)
- WakeGenerator
 - Triton::WakeGenerator, [146](#)
- WakeGeneratorParameters
 - Triton::WakeGeneratorParameters, [150](#)
- WindFetch
 - Triton::WindFetch, [153](#)
- x
 - Triton::Vector3, [137](#)
 - Triton::Vector3f, [140](#)
 - Triton::Vector4, [143](#)